CSIG USER MANUAL

Introduction

CSIG (Chen Sen Integrator) is an integrator product which can interpret vendor protocols (e.g.: Modbus RTU...) into another protocols (e.g.: Johnson Controls N2, BACnet MS/TP, Modbus RTU…).

Overview

CSIG hardware includes:

1. CPU board: An Emtronix ETR232I board. It includes two RS232 ports and one Ethernet port. CSIG used 2 of the RS232 ports.
2. LCD board: A LCD board which can link with CPU board. It will display the error message which is very useful for new user.

CSIG software includes: The software listed below is the software that has been downloaded to CPU board

1. USER.BAT: The batch file which let CPU board know which exe file will execute.
2. CSIG.EXE: CSIG main software.
3. SITEKEY.CSV: The file includes sitekey. Without site key, the CSIG can work with full functions for 10 days. After that, the CSIG will stop working and need to re-start by push reset button.
4. CSIG.INI: CSIG settings, such as protocol, port and so on.
5. CSIGCFG.CSV: CSIG main configuration file. To define the linkage between devices.
6. TABLE0.CSV: Table file for interface protocol. CSIG can define and download 16 table files. TABLE0.CSV to TABLE15.CSV. For most application, one table file (TABLE0.CSV) is enough.

The software files listed above have been downloaded to CPU board by factory setup. For modification on configuration files, user can use WTDRF to communicate with CSIG, and normally only need to change and download the following files:

1. CSIGCFG.CSV
2. TABLE0.CSV
3. CSIG.INI

Capacity

- Slave protocol supported: Johnson Controls' N2, BACnet MS/TP or Modbus (slave).
- Slave port supported: 1
- Vendor RS232 port supported: 1
- Vendor protocol supported: ANY RS232/485 protocol
- Maximum N2 devices supported per port: 50
- Maximum BACnet MS/TP devices supported per port: 32

- Maximum Modbus devices supported per port: 32 (Note 1)
- Maximum types of table per port: 16 (Note 2)
- Points per N2 device:
  - AI: 255
  - BI: 255
  - AO: 255
  - BO: 255
- Points per BACnet MS/TP device:
  - AI: No limitation
  - BI: No limitation
  - AO: No limitation
  - BO: No limitation
  - The points' number is restricted by hardware resource.
- Points per Modbus RTU (slave) device:
  - BI: No limitation
  - The points' number is restricted by hardware resource.
  - Present version CSIG can only support function 2 "Read Input Status"

Note 1: Maximum Modbus devices supported:
Logically you can address over 250 devices; however, the RS-485 transceivers are not capable of physically driving that many devices. Modbus protocol states that the limit is 32 devices, and most RS-485 transceivers will agree with this. Only if all devices on the network have low load transceivers can you have more than 32 devices.

Note 2: Maximum types of table:
You can connect different devices on the same RS-485 bus, as far as they have the same physical property. For example, you can connect different Modbus RTU devices into one CSIG. Each vendor device will have different mapping table. The maximum different mapping table you can have is 16.

Note 3: Maximum IO Points supported:
Since the CSIG is acted as a Protocol Converter. There is NO limitation on the total number of points supported. However, there is a limitation on the Johnson Controls' Supervisory Controllers:

Supervisory Controller	Max IO Point Support
NCM350-8	800
N30	500
NCE25	500
NAE35	500
NAE45	1,000
NAE55	2,000

Installation Procedures

Figure 1. CSIG

Installation

1. Connect +5V DC power supply (included) to CSIG and **RS232/RS485 Converter**.
2. Use RS232 cable or RS485/RS232 converter to link vendor device to COM1.
3. Set jumper on JP2 to RS485 (default).
4. Link N2, BACnet MS/TP or Modbus RTU (slave) cable to RS-485.
5. Check there is no jumper on JP1 and no jumper on CN10 pin15-16.

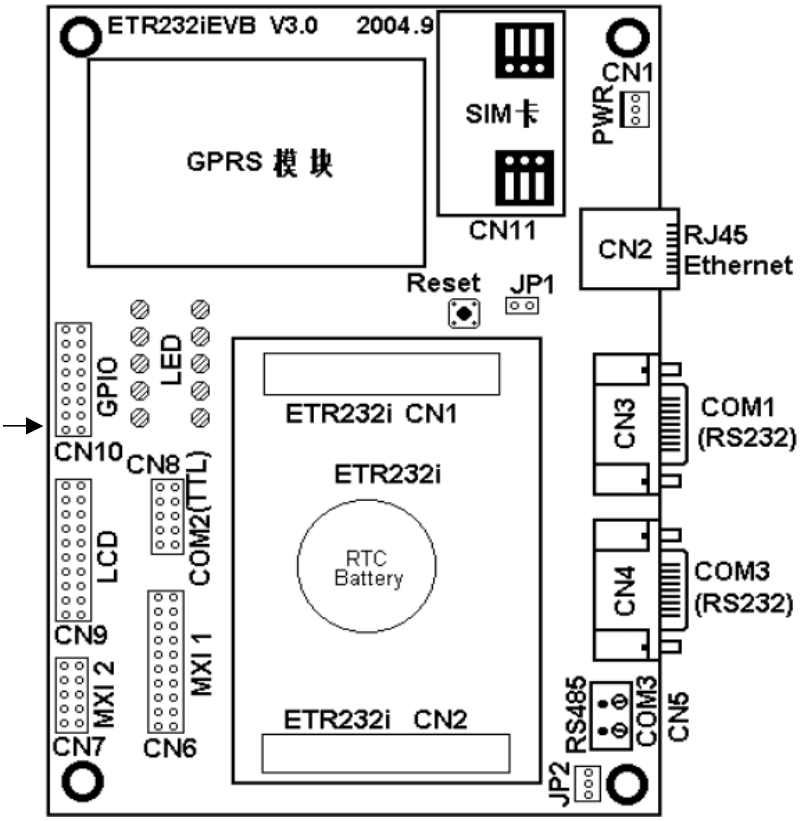

Figure 2. CSIG Board Layout

Operation Procedures

During operation, a CSIG has 2 modes: Operation Mode, and Download Mode:

Operation Mode: CSIG can establish communication with both vendor device and N2 host. It's the working mode in routine operation.

Download Mode: User can download files to CSIG by WTDRF for configuring or upgrading. During this time, the CSIG will stop polling and cannot get Slave online.

Operation Mode

In this mode, the LCD should display:

CSIG ver 3.x
IP: 192.168.211.211
License: Yes

After delay 3 seconds, the LCD will display communication message, and error message if any.

Message	Description
Tx: xxxxxxxxxx	The communication stream which CSIG is sending to vendor device.
Rx: xxxxxxxxxx	The communication stream which CSIG received from vendor device

Operation mode is the most often used mode. In this mode, CSIG will implement integrator's functions, polling with vendor device and exchanging data with N2 host.

Download Mode

WTDRF.exe is a DOS software which can be used to download software to CSIG. It can work on standard RS232 port or USB RS232 port.

1. Installation: Create a working directory "c:\wtdrf", and copy the wtdrf.exe into it; Put Pcomm.dll in c:\windows\system32, and add the wtdrf.exe path in autoexec.bat. E.g.:

   PATH = %PATH% c:\wtdrf

   This will allow the user to use wtdrf without typing path.

2. Link RS232 cross cable to link the PC and CISG COM1. The RS232 cross cable is defined as follows:

   DB9 DB9
   RX 2----------3
   TX 3----------2
   GND 5----------5

3. Plug a jumper on JP1.
4. Push reset button, and the LCD board should display: "Waiting for hand shake"
5. Use the command as follows to download/upload files.

Command	Function
wtdrf –rp# d	Display a list of files in CSIG
wtdrf –rp# e filename	Delete a file in CSIG
wtdrf –rp# t filename	Copy a file TO CSIG
wtdrf –rp# f filename	Copy a file FROM CSIG

Note: -rp# is the port number for the PC. E.g.: If port 1 is used, the above command will be implemented as:
wtdrf –rp1 d : display the file list on the CSIG
wtdrf –rp1 e table0.csv : delete the file table0.csv
wtdrf –rp1 t csigcfg.csv: copy the file csigcfg.csv from PC to CSIG
wtdrf –rp1 f csig.ini : copy the file csig.ini from CSIG to PC

6. After finishing, unplug the jumper on JP1, push the reset button and the CSIG will enter operation mode.

Configuration

CSIG configuration includes 2 types of csv files modification and downloading on 2 types of csv files.

File Name	Description
CSIG.INI	Define slave protocol type (N2, BACnet MS/TP or Modbus RTU slave), and other parameters.
CSIGCFG.CSV	The CSIG configuration which includes N2/BACnet/ Modbus slave address, vendor address, and so on
TABLE0.CSV	The protocol definition files for vendor device communication. It includes all of AI, BI, AO and BO definition.
table1.csv, table2.csv… table15.csv	Same as table0.csv. CSIG can support 16 different tables (Vendor Family).

Normally, TABLE0.CSV can be obtained from CSIG table library or ordered from Chen Sen Controls Ltd if it's a special application. The user needs to modify CSIGCFG.CSV file by him/her. It is recommended that the user use CSIG sample Excel file to modify and save as csv file. That can reduce the mistake.

TABLE0.CSV

CSIG can define and download 16 tables. The name must be Table0.csv to Table15.csv. In a working CSIG, it must include at least 1 table file (table0.csv).

Guidelines for string definition in table definition:

1. An unprintable ASCII code can be expressed as a Hex decimal with single quotes. E.g.: '0d' is ASCII CR.
2. A printable ASCII can also be expressed in hex decimal. E.g.: '32' is ASCII 2. The content in quotes is case insensitive. E.g.: '0d' is the same as '0D'.
3. In one pare of quotes, only ASCII code can be defined, no more, no less. E.g.: '0d0a' is illegal, '0d''0a' is a right expression.

<TYPE> <END> block

Field Name	Description
Table Name	The name of this protocol definition. This name will be used in CSIGCFG.csv file. E.g.: GE PLC
Protocol Type	In current CSIG, 4 protocol types are supported. They are: 1. BINARY: For general Binary MASTER/SLAVE protocol. The CRC can be of many types. The response won't check CRC except CH01 (ModBus RTU CRC16). 2. ASCII: For general ASCII MASTER/SLAVE protocol. 3. BinEvent: For Binary Event protocol. In this protocol, CSIG won't send poll out, only listen to the incoming message, and update Ai/Bi accordingly.

	4. ASCEvent: For ASCII Event protocol. In this protocol, CSIG won't send poll out, only listen to the incoming message, and update Ai/Bi accordingly.
Table Desp	Description of the table

<AI> <END> block

Field Name	Description
Name	N2 point name. It must start with ai1, without any skip. E.g.: ai1, ai2, ai3, ai4 is a legal definition. But ai1, ai3, ai4 is a wrong definition since it skips ai2.
Description	Point definition
Poll	This data is got from which polling. This polling name must match one of the poll name in <POLL><END> block
Data Size	The data is composed by how may bytes.
Position	The start position of data in response stream.
Convert type	How to convert the vendor data into N2 data. Please refer to Appendix for the definition of the Convert type.
a (coefficient)	If vendor value = f, N2 AI value = a*f +b
b (offset)	If vendor value = f, N2 AI value = a*f +b
Validation	Validation string. In certain conversion type (e.g. C102), the CSIG uses validation string to location the value position.

<BI> <END> block

Field Name	Description
Name	N2 point name. It must start with bi1, without any skip. E.g.: bi1, bi2, bi3, bi4 is a legal definition. But bi1, bi3, bi4 is a wrong definition since it skips bi2.
Description	Point definition
Poll	This data is got from which polling. This polling name must match one of the poll name in <POLL><END> block
Data Size	The data is composed by how may bytes.
Position	The data position in response stream.
Bit posi/or Byte Val	The BI is defined in which bit in the byte. E.g.: if Data position =3, bit poi =4, that means this BI is corresponding to 3rd byte, 4th bit of the response stream. In Cbyte type, this field is for the byte value. E.g.: if Data start position = 3, bit posi/byte Val = 123, that means this BI is True if 3rd byte value is 123.
Convert type	How to convert the vendor data into N2 data. Please refer to Appendix for the definition of the Convert type.
Validation	Validation string. In certain conversion type (e.g. C102), the CSIG uses validation string to location the value position.
Value True	The string indicates the true value. In certain conversion type. (e.g. C102), CSIG check the response string. If the string match the "Value True" (E.g.: response START, if the value could be START/OFF), the Bi will be set TRUE. Otherwise sets False

<AO> <END> block

Field Name	Description
Name	N2 point name. It must start with ao1, without any skip. E.g.: ao1, ao2, ao3, ao4 is a legal definition. But ao1, ao3, ao4 is a wrong definition since it skips ao2.
Description	Point definition
Poll	This data is got from which polling. This polling name must match one of the poll name in <POLL><END> block
Data Size	The data is composed by how may bytes.
Position	The start position of data in response stream.
Convert type (Read)	How to convert the vendor data into N2 data. Please refer to Appendix for the definition of the Convert type.
a (coefficient)	If vendor value = f, N2 AI value = a*f +b
b (offset)	If vendor value = f, N2 AI value = a*f +b
Validation	Validation string. In certain conversion (e.g. C102), the CSIG uses validation string to location the value position.
Poll (Write)	To define the poll used to write the N2 data to vendor device. This polling name must match one of the poll name in <POLLW><END> block
Convert type (Write)	How to convert the N2 data into vendor data. Please refer to Appendix for the definition of the Convert type.

<BO> <END> block

Field Name	Description
Name	N2 point name. It must start with bo1, without any skip. E.g.: bo1, bo2, bo3, bo4 is a legal definition. But bo1, bo3, bo4 is a wrong definition since it skips bo2.
Description	Point definition
Poll	This data is got from which polling. This polling name must match one of the poll name in <POLL><END> block
Data Size	The data is composed by how may bytes.
Position	The data position in response stream.
Bit posi/or Byte Val	The BI is defined in which bit in the byte. E.g.: if Data position =3, bit poi =4, that means this BI is corresponding to 3rd byte, 4th bit of the response stream. In Cbyte type, this field is for the byte value. E.g.: if Data start position = 3, bit posi/byte Val = 123, that means this BI is True if 3rd byte value is 123.
Convert type (Read)	How to convert the vendor data into N2 data. Please refer to Appendix for the definition of the Convert type.
Validation	Validation string. In certain conversion type (e.g. C102), the CSIG uses validation string to location the value position.
Value True	The string indicates the true value. In certain conversion type. (e.g. C102), CSIG check the response string. If the string match the “Value True” (E.g.: response START, if the value could be START/OFF), the Bi will be set TRUE. Otherwise False
Poll (Write)	To define the poll used to write the N2 data to

	vendor device. This polling name must match one of the poll name in <POLLW><END> block
Convert type (Write)	How to convert the N2 data into vendor data. Please refer to Appendix for the definition of the Convert type.

<POLL> <END> block: For poll reading.

<POLLW> <END> block: For poll writing.

Field Name	Description
Name	For <POLL> Poll name. It must start from p00 and in continues. E.g.: p00, p01, p02 are 3 legal polls. But p00, p01, p03 will give error since there is short of p02 between p01 and p03. For<POLLW> Same guideline, but the poll name should be wp00, wp01, wp02…
POLL	Poll definition: 1**. for Binary table type**. E.g.: addr 060002 data crc. “addr” is the part to be replaced with a vendor address in real poll. “data” is the part to be replaced with the data in real poll. “crc” is the part to be replaced with CRC16 (or other checksum algorithm). Please refer to the <SEED> block and appendix for the addr/data/crc details. For binary table type (Modbus_RTU), CSIG will take out space in the poll definition before sending send real poll, that is “01 02 03” is the same as “010203”. This can make the poll definition more readable. 2. **for ASCII or ASCEvent table type** E.g.: WR aa ANALOG_OUT AO1 dd’0d’ “aa” is the part to be replaced with a vendor address in real poll. “dd” is the part to be replaced with the data in real poll. Please refer to the <SEED> block and appendix for the aa/dd details.
Resp bytes or end msg	1. If fill in only a number, CSIG recognizes it as “BY_LENGTH”. A successful respond is a fixed number of character . E.g.: 10, means wait for 10 bytes as a legal response. If the vendor response bytes got by CSIG is more than or less than this number, it will be considered as a wrong response. 2. If fill in a string started with “w4+” and followed with a substring, CSIG recognizes the substring as “BY_ENDMSG”. E.g.: w4+END, means wait until get the string “END”, and then consider the response as a legal response. 3. If fill in a string started with “w4>” and followed with a number, CSIG recognizes the substring as “BY_TIMEOUT”. The poll will wait until time out, and then check whether the response number of chars is more than designed number. If yes, the response is OK, if not, the response failed. 4.If fill in a string with 2 digit separated with a “:”, CSIG recognizes it as “BY_VALID”. E.g.: “80:75” means the respond is 80 characters with the validation string at position 75. The user must define validation string.
Validation	If fill in a string: 1. CSIG will double check the response for this validation string before considers the response

	as a legal one. 2. Some of the convert type will use validation string as the start of “Position” counting. If leave it as blank, no validation.
Value true	The value that the software recognizes as Boolean TRUE. (only useful for <POOLW>)
Value false	The value that the software recognizes as Boolean FALSE. (only useful for <POOLW>)
Comments	Comments from user, e.g.: the calculation of response bytes. No use to the CSIG, only for user’s reference.

<SEED> <END> block

Seed block is to define the addr/data/checksum, and other string (call seed) which is used in poll definition for replacement with real values in real communication.

Field Name	Description
Name	Seed name
String in poll	The string represents seed in the poll string. CSIG will search this string in poll, and recognizes the string as seed.
Type	Seed type. Please refer to the Appendix for the type definition.

<ENUM> <END> block

Enum block is to define the enumric data type. It is used to convert a string into a pre-defined integer. E.g.: “NORMAL” = 0, “TROUBL”=1, “ALARM:”=2.

Field Name	Description
Name	ASCII string
Value	A integer

CSIGCFG.CSV

Field Name	Description
N2/BACnet/Modbus slave Address	N2/BACnet address on which one vendor device will be mapped. For N2, address range is 1-255. For BACnet MS/TP, address range is 4-127. For Modbus slave, address rang is 1-254.
Table name	The protocol definition table name, it must match the table name defined in TABLE0.CSV
Vendor Address	1-255, or others based on the protocol definition
Port	The com port that the vendor device will link to. In current CSIG version, only PORT_A can be used. The spelling “PORT_A” must right.
Time out (ms)	Polling time out. E.g.: 1000
Scan dly (ms)	The time delay between two-success polling. E.g.: 500
Retry	The retry of the polling if polling fails. 0 is no retry or 1 polling; 1 is one retry, or 2 poling. E.g.: 2
Port Name	CSIG internal port name definition. Users should not modify this field.
Port Def	Define which com port will be assigned to the port name. Users should not modify this field.
Baud Rate	2400,4800,9600,19200,38400. For N2 slave protocol: PORT_N2 baud rate should be 9600 (not adjustable). For BACnet MS/TP slave protocol: PORT_N2 baud rate should be 38400 (adjustable). For Modbus RTU slave: PORT_N2 baud rate should be 9600 (adjustable).
Data bits	7, 8 for 7, 8 data bits
Parity	N,E,O for None, Even, Odd
Stop bits	In current CSIG version, only 1 can be set

CSIG.INI

Field Name	Description
[VENDOR_PORT]	**Please keep the content as:** Vendor_Port=PORT_A
[Ethernet]	**Please keep the content as:** LocalIP=192.168.211.211 Subnet_mask=255.255.255.0 GateWayIP=0.0.0.0
[FTP]	**Please keep the content as:** Timeout=30000 UserID=admin Password=1234

[PROTOCOL]	Slave protocol type, and max master address (for BACnet MS/TP)
	
	If the slave protocol is N2, configure as follows:
	[PROTOCOL]
	Protocol_Type=N2
	//Protocol_Type=MODBUS
	//Protocol_Type=BACNET
	
	If the slave protocol is BACnet MS/TP, configure as follows:
	[PROTOCOL]
	//Protocol_Type=N2
	//Protocol_Type=MODBUS
	Protocol_Type=BACNET
	Max_Master=10
	
	If the slave protocol is Modbus RTU slave configure as follows:
	[PROTOCOL]
	//Protocol_Type=N2
	Protocol_Type=MODBUS
	//Protocol_Type=BACNET

N2 definition

CSIG should be configured as the VND type in Metasys N2.

Appendix

Protocol Type Definition

1. **Binary:** It's a binary string.

2. **ASCII:** It's an ASCII string.

3. **ASCEvent/BINEvent:** For ASCII/Binary Event protocol. In this protocol, CSIG won't send polling out, only listen to the incoming message, and update point accordingly. ASCEvent is for the incoming message that is ASCII string. BINEvent is for the incoming message that is binary string. At present CSIG version, ASCEvent has the limitations as follows:

- Can not work together with other type of protocol
- Can only support AI and BI. Can support multi slave devices.
- In CSIGCFG.csv, recommend to set "Time out(ms)" =1000 to 5000; "Poll dly(ms)" = 0; "Retry" = 0.
- Each of the poll must define validation, since the ASCEvent/BINEvent is using validation to get the current poll.

Conversion Type (Read) Definition

Below are the conversion types (read) supported by the present version of CSIG and their definitions in C++ code.

Convert Type (Read)

Type Name	Description	Data Size	Protocol type
C01	Unsigned	1,2,4	BINARY,BinEvent . ai,ao
C02	Unsigned swap	1,2,4	BINARY,BinEvent ai,ao
C03	Signed	1,2,4	BINARY,BinEvent ai,ao
C04	Signed swap	1,2,4	BINARY,BinEvent ai,ao
C05	IEEE 32 bit float	4	BINARY,BinEvent ai,ao
C06	IEEE 32 bit float swap	4	BINARY,BinEvent ai,ao
C07	32 bit unsigned byte swap, no word swap	4	BINARY,BinEvent ai,ao
C08	32bit signed byte swap, no word swap	4	BINARY,BinEvent ai,ao
C09	signed byte swap, word not swap. E.g: 1DC0FFFE = -123456	4	BINARY,BinEvent. ai, ao
C10	2 bytes. High = int (0-99), Low=decimal (0/5). E.g.: 25.5=1905,H 5.0=0500H	2	BINARY, BinEvent. ai,ao
C11	IEEE floating in Modbus. E.g.: 615B3F6C = 0.9234	4	BINARY, BinEvent. ai,ao
C12	Signed swap : MSB is the sign E.g.: 8123H= -291	2	BINARY, BinEvent ai, ao
C20	1 word is a data ranged from 0—10000.E.g.: E.g.: 04D2162E = 12345678	4,6,8	BINARY, BinEvent ai,ao

C21	1 word is a data ranged from 0—10000, but swap E.g: 04D2162E = 56781234	4,6,8	BINARY, BinEvent ai ao
C22	Signed modulus 10000 E.g.: FF439EB2 =-1234*10000+-5678 = -12345678	4,6,8	BINARY, BinEvent ai ao
CBitwise	bitwise unsigned. E.g.: 10010001, bit0-4. ai = 17	1, 2 (bit0-15)	BINARY,BinEvent ai,ao,
CBit	Bi get value by bitwise	1,2	BINARY,BinEvent bi,bo
CByte	Bi get value by comparing byte value	1	BINARY,BinEvent bi,bo
CTF	Direct read True/False from head of the msg. True/False values is defined in poll.	n.a.	ASCII, ASCEvent bi, bo.
C101	ASCII direct conversion: Data offset from start of msg. e.g.: "12.3"->12.3, “Start”->True (if “Value true” = Start)	n.a.	ASCII, ASCEvent. ai, ao, bi, bo.
C102	ASCII direct conversion: Data offset from validation end.	n.a.	ASCII, ASCEvent. ai, ao, bi, bo.
CEnum1	ENUM (a set of pre-defined value) type: Data offset from start of msg. E.g.: “Normal”=0, “Troubl”=1, “Alarm:”=2.	n.a. string	ASCII, ASCEvent ai, ao
CEnum2	ENUM type: Data offset from validation end	n.a. string	ASCII, ASCEvent ai, ao
CBCD	BCD normal: 12H = 12f ,1234H=1234f, 12345678H = 12345678f	1,2,4 or others	BINARY,BinEvent ai, ao

Below is the detail implementation of the conversion types:

```
switch (cvttype_r)
{
  case C01:  // unsigned  E.g.: 40E20100 = 123456
          switch (datasize)
          {

            case 1:
                f=(float) *((unsigned char *) (msg+byteposi));
                break;
            case 2:
                f=(float) *((unsigned int *) (msg+byteposi));
                break;
            case 4:
                f= (float) *((unsigned long *) (msg+byteposi));
                break;
            default: WarnLog("No such datasize", DIS_ALL);
          }
          break;

  case C02:  // unsigned swap E.g: 0001E240 = 123456
          switch (datasize)
          {
            case 1:
                f=(float) *((unsigned char *) (msg+byteposi)); // same as no swap
                break;
            case 2:
                c[0] = *(msg+byteposi+1);
                c[1] = *(msg+byteposi);
                f=(float) *((unsigned int *) (c));
                break;
            case 4:
                c[0] = *(msg+byteposi+3);
```

```
                c[1] = *(msg+byteposi+2);
                c[2] = *(msg+byteposi+1);
                c[3] = *(msg+byteposi);
                f= (float) *((unsigned long *) (c));
                break;
          default: WarnLog("No such datasize",DIS_ALL);
        }
        break;

case C03:  // signed.     E.g: C01DFEFF = -123456
        switch (datasize)
        {
          case 1:
                f=(float) *((char *) (msg+byteposi));
                break;
          case 2:
                f=(float) *((int *) (msg+byteposi));
                break;
          case 4:
                f= (float) *((long *) (msg+byteposi));
                break;
          default: WarnLog("No such datasize", DIS_ALL);
        }
        break;
case C04:  // signed swap   e.g: FFFE1DC0 = -123456
        switch (datasize)
        {
          case 1:
                f=(float) *((char *) (msg+byteposi)); // same as no swap
                break;
          case 2:
                c[0] = *(msg+byteposi+1);
                c[1] = *(msg+byteposi);
                f=(float) *((int *) (c));
                break;
          case 4:
                c[0] = *(msg+byteposi+3);
                c[1] = *(msg+byteposi+2);
                c[2] = *(msg+byteposi+1);
                c[3] = *(msg+byteposi);
                f= (float) *((long *) (c));
                break;
          default: WarnLog("No such datasize !", DIS_ALL);
        }
        break;

case C05:  // IEEE 32-bits E.g. 0020F1C7= -123456
        f= *((float *) (msg+byteposi));
        break;
case C06:  // IEEE 32-bit swap. E.g. C7F12000= -123456
        c[0] = *(msg+byteposi+3);
        c[1] = *(msg+byteposi+2);
        c[2] = *(msg+byteposi+1);
        c[3] = *(msg+byteposi);
        f= *((float *) (c));
        break;
case C07: //unsigned byte swap, word not swap. E.g:E2400001 = 123456
        switch (datasize)
        {
          case 4:
                c[0] = *(msg+byteposi+1);
                c[1] = *(msg+byteposi+0);
                c[2] = *(msg+byteposi+3);
                c[3] = *(msg+byteposi+2);
                f= (float) *((unsigned long *) (c));
                break;
          default: WarnLog("No such datasize !", DIS_ALL);
        }
        break;
case C08: //signed byte swap, word not swap. E.g: 1DC0FFFE = -123456
        switch (datasize)
        {
          case 4:
                c[0] = *(msg+byteposi+1);
```

```
                    c[1] = *(msg+byteposi+0);
                    c[2] = *(msg+byteposi+3);
                    c[3] = *(msg+byteposi+2);
                    f= (float) *((long *) (c));
                    break;
                default: WarnLog("No such datasize !", DIS_ALL);
                }
                break;
        case C11:  // IEEE 32-bit (modbus. E.g: 615B3F6C = 0.9234
                c[0] = *(msg+byteposi+1);
                c[1] = *(msg+byteposi+0);
                c[2] = *(msg+byteposi+3);
                c[3] = *(msg+byteposi+2);
                f= *((float *) (c));
                break;
        case C12: // signed swap 16bit, MSB is sign. E.g: -31794=974
                switch (datasize)
                {
                  case 2:
                    c[0] = *(msg+byteposi+1);
                    c[1] = *(msg+byteposi+0);
                    if ((c[1] & 0x80) == 0) {f1 = 1;}
                    else {f1 = -1;}
                    c[1] = c[1] & 0x7F;
                    f= (float) *((int *) (c));
                    f = f*f1;
                    break;
                  default: WarnLog("No such datasize !", DIS_ALL);
                }
                break;
      case C20: // same as MIG RD21. E.g: 04D2162E = 12345678
          //  1234=04D2H  5678=162EH  9012=2334H  3456=0D80H
          //  {0x04, 0xd2, 0x16, 0x2e, 0x23, 0x34, 0x0d, 0x80};
          // result = 1234567890123456.

                switch (datasize)
                {
                  case 4:
                  case 6:
                  case 8:
                    n = datasize/2;
                    dbf = 0;
                    for (i=0; i<n; i++)
                    {
                      c[0] = *(msg+byteposi+i*2+1);
                      c[1] = *(msg+byteposi+i*2+0);
                      ff[i] = (float) *((unsigned int*) (c));
                      dbf = dbf*10000+ff[i];
                    }
                    f = dbf;
                    break;
                  default: WarnLog("No such datasize !", DIS_ALL);
                }
                break;
        case C21: // same as MIG RD21. E.g: 04D2162E = 56781234
          //  1234=04D2H  5678=162EH  9012=2334H  3456=0D80H
          //  {0x04, 0xd2, 0x16, 0x2e, 0x23, 0x34, 0x0d, 0x80};
          // result = 3456901256781234.

                switch (datasize)
                {
                  case 4:
                  case 6:
                  case 8:
                    n = datasize/2;
                    dbf = 0;
                    for (i=0; i<n; i++)
                    {
                      c[0] = *(msg+byteposi+(n-1-i)*2+1);
                      c[1] = *(msg+byteposi+(n-1-i)*2+0);
                      ff[i] = (float) *((unsigned int*) (c));
                      dbf = dbf*10000+ff[i];
                    }
                    f = dbf;
```

```
                    break;
                default: WarnLog("No such datasize !", DIS_ALL);
            }
            break;

case C22: // 32 bit modulus-10000 signed format.
   // it breaks a 32 bit value into two 16 bit registers.
   // the value = reg_high*10000 + reg_low
   // e.g: reg_high = -1234 = FB2E, reg_low =-5678=E9D2
   // value = -1234*10000+(-5678) = - 12345678
                        switch (datasize)
                        {
                          case 4:
                          case 6:
                          case 8:
                                n = datasize/2;
                                dbf = 0;
                                for (i=0; i<n; i++)
                                {
                                  c[0] = *(msg+byteposi+i*2+1);
                                  c[1] = *(msg+byteposi+i*2+0);
                                  ff[i] = (float) *((signed int*) (c));
                                  dbf = dbf*10000+ff[i];
                                }
                                f = dbf;
                                break;
                          default: WarnLog("No such datasize !");
                        }
            break;

    case CBITWISE: // bitwise unsigned. Several bits in one byte. E.g.: 10010011, bit0-4 = 3
            d = *((unsigned int*)(msg+byteposi));
            c1 = 0xFFFF;
            if (bitp2 < bitp1) return 0;
            c1 = c1 << (bitp2-bitp1+1);
            c1 = ~ c1;
            d = d >> bitp1;
            d = d & c1;
            f=(float) d;
            break;
    case CBCD:  // BCD normal E.g.: 12h=12f, 1234h = 1234f, 12345678h= 12345678f
            f = BCD2Float(msg+byteposi, datasize); // any size can handle
            break;
    default:   WarnLog("No such datatype!",DIS_ALL);
            break;
 }
 f = a*f + b;
-----------------------------------------------------------------------------------------------------------------
switch (cvttype_r)
 {
   case C101: // ASCII direct convert, position from string start. E.g. “-12345.6” = -12345.6
            strxfrm(str, msg+byteposi, datasize+1);
            break;
   case C102: // ASCII direct convert, position start from validation end. E.g. “-12345.6” = -
12345.6

            p1 = strstr(msg, strvalid);
            if (p1 == NULL) return 0;
            strxfrm(str, p1+strlen(strvalid)+byteposi, datasize+1);
            break;
   default:        return 0;
 }
 f = atof(str);
 f = a*f + b;
-----------------------------------------------------------------------------------------------------------------
 switch (cvttype_r)
 {
   case CBIT: // Bi get value by bitwise.
         if (*(msg+byteposi) & bitValue[bitposi]) return TRUE;
         else return FALSE;
   case CBYTE: // Bi get value by comparing byte value
         return ((*(msg+byteposi) == cbyte)? TRUE:FALSE);
   case CTF:   // Direct read True/False, from head of string
         return bincmp1(msg+byteposi, datasize
```

```
                , strTrue+1, *(strTrue+0));
  default:
        ErrLog("No such cvt type in getb_mod");
        return FALSE; // only clear compile warning
 }
```

Conversion Type (Write) Definition

Below are the conversion types (write) supported by the present version of CSIG and their definitions in C++ code.

Convert Type (Write)

Type Name	Description	Data Size	Protocol type
CW01	Unsigned	1,2,4	BINARY ao
CW02	Unsigned swap	1,2,4	BINARY ao
CW03	Signed	1,2,4	BINARY ao
CW04	Signed swap	1,2,4	BINARY ao
CW05	IEEE 32 bit float	4	BINARY ao
CW06	IEEE 32 bit float swap	4	BINARY ao
CW07	32bit unsigned byte swap, no word swap	4	BINARY ao
CW08	32bit signed byte swap, no word swap	4	BINARY ao
CW11	IEEE floating in Modbus. E.g.: 615B3F6C = 0.9234	4	BINARY ao
CWBit	Write value by bitwise	1,2	BINARY bo
CW	Direct data writing. Replace the seed “data” in pollw directly with the ASCII value string (analog) or valuetrue/valuefalse defined in pollw. E.g.: WV Ao1 data → WV Ao1 23.4 WV Bo1 data → WV Bo1 start Here start is the valuetrue defined in pollw.	n.a.	BINARY bo. ASCII ao, bo
CWBCD	BCD normal. 12345678f = 12345678H.	1,2,4 or others	BINARY ao

```
---------------------------------------------------------------------------------------------------------
// BINARY AO
  switch (cvttype)
  {
     case CW01:  // unsigned  E.g.: 40E20100 = 123456
              str = fcvt(fvalue, 0, &dec, &sign);
              ul =strtoul(str, &endptr, 10);
              switch (datasize)
              {
                case 1:
                     msg[1] = *((char*) &ul);
                     msg[0] = 1;
                     break;
                case 2:
                     *((unsigned*) &msg[1]) =*((unsigned*) &ul);
                     msg[0] = 2;
                     break;
                case 4:
                     *((unsigned long*) &msg[1]) = ul;
                     msg[0] = 4;
```

```
            break;
        default: WarnLog("No such datasize", DIS_ALL);
                 datasize = 0;
      }
      break;

case CW02:  // unsigned swap E.g: 0001E240 = 123456
      str = fcvt(fvalue, 0, &dec, &sign);
      ul =strtoul(str, &endptr, 10);
      switch (datasize)
      {
        case 1:
            msg[1] = *((char*) &ul);
            msg[0] = 1;
            break;
        case 2:
            *((unsigned*) &msg[1]) =*((unsigned*) &ul);
            swap(msg+1,2);
            msg[0] = 2;
            break;
        case 4:
            *((unsigned long*) &msg[1]) = ul;
            swap(msg+1,4);
            msg[0] = 4;
            break;
        default: WarnLog("No such datasize", DIS_ALL);
                 datasize = 0;
      }
      break;

case CW03:  // signed. E.g: C01DFEFF = -123456
      str = fcvt(fvalue, 0, &dec, &sign);
      if (sign != 0) // add sign.
      { strcpy(c, "-");
        strcat(c, str);
      }
      else
      { strcpy(c, str);}

      switch (datasize)
      {
        case 1:
            i = atoi(c);
            *((char*) &msg[1]) = (char) i;
            msg[0] = 1;
            break;
        case 2:
            *((int*) &msg[1]) = atoi(c);
            msg[0] = 2;
            break;
        case 4:
            *((long*) &msg[1]) = atol(c);
            msg[0] = 4;
            break;
        default: WarnLog("No such datasize", DIS_ALL);
                 datasize = 0;
      }
      break;
case CW04:  // signed swap E.g: FFFE1DC0 = -123456
      str = fcvt(fvalue, 0, &dec, &sign);
      if (sign != 0) // add sign.
      { strcpy(c, "-");
        strcat(c, str);
      }
      else
      { strcpy(c, str);}

      switch (datasize)
      {
        case 1:
            i = atoi(c);
            *((char*) &msg[1]) = (char) i;
            msg[0] = 1;
            break;
```

```
                case 2:
                    *((int*) &msg[1]) = atoi(c);
                    swap(msg+1,2);
                    msg[0] = 2;
                    break;
                case 4:
                    *((long*) &msg[1]) = atol(c);
                    swap(msg+1,4);
                    msg[0] = 4;
                    break;
                default: WarnLog("No such datasize !", DIS_ALL);
                datasize = 0;
            }
            break;

    case CW05:  // IEEE 32-bits E.g: 0020F1C7= -123456
            *((float*) (msg+1)) = fvalue;
            msg[0] = 4;
            break;
    case CW06:  // IEEE 32-bit swap E.g: C7F12000= -123456
            *((float*) (msg+1)) = fvalue;
            swap(msg+1,4);
            msg[0] = 4;
            break;
    case CW07: //unsigned byte swap, word not swap. E.g: E2400001 = 123456
            str = fcvt(fvalue, 0, &dec, &sign);
            ul =strtoul(str, &endptr, 10);
            switch (datasize)
            {
              case 4:
                    *((unsigned long*) &msg[1]) = ul;
                    swap(msg+1,2);
                    swap(msg+1+2,2);
                    msg[0] = 4;
                    break;
              default: WarnLog("No such datasize", DIS_ALL);
            }
            break;
    case CW08: //signed byte swap, word not swap. E.g: 1DC0FFFE = -123456
            str = fcvt(fvalue, 0, &dec, &sign);
            if (sign != 0) // add sign.
            { strcpy(c, "-");
              strcat(c, str);
            }
            else
            { strcpy(c, str);}

            switch (datasize)
            {
              case 4:
                    *((long*) &msg[1]) = atol(c);
                    swap(msg+1,2);
                    swap(msg+1+2,2);
                    msg[0] = 4;
                    break;
              default: WarnLog("No such datasize", DIS_ALL);
            }
            break;
    case CW11:  // IEEE 32-bit modbus swap E.g: 615B3F6C= 0.9234
            *((float*) (msg+1)) = fvalue;
            swap(msg+1,2);
            swap(msg+3,2);
            msg[0] = 4;
            break;

    case CWBCD:// 12345678f to 12345678H
            Float2BCD((msg+1), fvalue, datasize);
            msg[0] = datasize;
            break;
    default:   WarnLog("No such datatype!", DIS_ALL);
            break;
-----------------------------------------------------------------------------------------------------------
// BINARY BO
  switch(cvttype)
```

```
{
  case CWBIT: // Binary bitwise setting. Modbus func 6/16. special handle in n2device
            *(msg+1) = *((char*) &bvalue + 1);
            *(msg+2) = *((char*) &bvalue);  // swap
            *(msg+0) = 2;
            break;
  case CW:     str = ((bvalue)? pp->strTrue : pp->strFalse);
            bincpy(msg, str);
            break;
  default:
            *((unsigned*) msg+1) = bvalue;
            *(msg+0) =2;
}

// ASCII AO
 switch ((m_pao+idx)->cvttype_w)
 {
    case CW:
    default:
          pp= (m_pw+(m_pao+idx)->idx_pw);
          strcpy(msg, pp->msg_proto);
          Seed2Real(msg, m_seed.strAddr, addr);
          sprintf(str, m_seed.strFmt, value);
          Seed2Real(msg, m_seed.strData, str);
          pp->len_msg = strlen(msg);
 }
 return pp;
-----------------------------------------------------------------------------------------------------------------
// ASCII BO.

 switch ((m_pbo+idx)->cvttype_w)
 {
    case CW:
    default:
          pp= (m_pw+(m_pbo+idx)->idx_pw);
          strcpy(msg, pp->msg_proto);
          Seed2Real(msg, m_seed.strAddr, addr);
          // value is a unsigned in Modbus, but a BOOL in others.
          if (value) {Seed2Real(msg, m_seed.strData, pp->strTrue);}
          else {Seed2Real(msg, m_seed.strData, pp->strFalse);}
          pp->len_msg = strlen(msg);
 }
 return pp;
```

Address Type (Seed)

Below are the conversion types (write) supported by the present version of CSIG and their definitions in C++ code.

Type Name	Description.
A01	1 byte binary address. e.g.: 01
A02	1-4 bytes Hex vendor address. E.g.: "12"→12H; "1234" → 1234H; "1234CD"→ 1234CDH; "123456CD"→123456CDH
A101	ASCII direct address. e.g.: "01"

Data Type (Seed)

Type Name	Description.
D101	ASCII direct data replacement.

Check Sum Type (Seed)

Type Name	Description.
CH01	Modbus RTU crc16
CH02	for ISOMAG Millennium 3 energy meter checksum 1. crc initial to zero. 2. crc loop rotated to the left by one bit 3. crc = crc + block bytes. E.g.: 11 FF 00 00 84; here 84 is the chksum.
CH03	Simple add: Just add all previous bytes together in one byte. E.g.: 55 AA 01 02 03 04 09; here 09 is the chksum
CH04	The 16 bit checksum, the 2's complement of the sum. E.g.:
CH05	Simple one byte check sum except first byte E.g.: 68 0B 00 01 3F 00 09 00 00 01 00 00 55 16 (55 is the check sum, add from 0B)

Format type (Seed)

Type Name	Description.
F101	ASCII c sprintf format. e.g.: "%f5.1" means length = 5, decimal position = 1