CorsairHMI Application Notes

9/18/2019

Contents

Browsing Architectures

Corsair can provide images that can be viewed with a browser. This browser can be running on a computer, a tablet, or a cell phone. Several architectures are possible.

Local Wifi

With this system the Corsair computer is tied by wireless or by a cable to a buildings Wifi router. The Corsair computer has a static IP address on the local network. This address must be compatible with the range of addresses used by the router. The computer must be set up to automatically log into the router.

Someone on the property with a cellphone or tablet must first connect to the wireless network. This requires the network name and password. After the tablet is logged in it's browser is pointed to the IP address of the Corsair computer.

This is a common architecture that only provides Corsair interface on the same property as the router. It does not permit access to Corsair over the internet, even if the router is hooked to the internet. Multiple Corsair computers can be hooked into the same router.

Wireless Router

A variation on the previous architecture is for the Corsair computer to act as the wireless router. This requires special software configuration of the computer's wireless adaptor. The tablet logs into the Corsair computer just as if it is logging into a separate router.

If there are several Corsair computers in a small area using this approach they must have different network names.

Internet Static IP

Many times it is desirable to browse into a Corsair computer from the Internet. The Corsair computer is hooked to a router that has internet access. The facilities Internet Service Provider (ISP) must provide a static known IP address for the facility. The router must be properly configured for port forwarding so that incoming browser request are sent to the Corsair computer.

One of the drawbacks to this approach is that some ISPs do not allow for static IPs. Many ISPs charge extra fees for a static IP. Configuring port forwarding on a router can be confusing with different equipment using different terminology. The local IP address on the Corsair computer must not change for the forwarding to continue working.

Push to the Cloud

Another approach is for the Corsair computer to 'push' data 'into the cloud'. Someone must rent space on a server computer in a data center. This server is accessed through a static known IP address that is provided by the data center. Corsair pushes data to the server computer. Cellphones can browse into the server computer using the static IP to see the data. Cellphones cannot browse directly into the Corsair computer.

One of the advantages of this architecture is that the Corsair computer does not have to have a static IP address as it initiates communications to the data center. The data center does not initiate communications to the Corsair computer. The Corsair computer can have a local IP address assigned automatically by a router. Multiple computers with similar configurations may peacefully coexist on the same router with this system.

One server computer could handle multiple Corsair computers. It could do extensive processing of the data. It could handle many more browsers being used at the same time than would be possible with browsing directly into an inexpensive Corsair computer. Data center Internet connections may be much faster than typical home connections.

One disadvantage of this approach is the cost of the data center's services. Another is that software must be purchased or developed for the server.

There are several methods that can be used for the Corsair computer to pass data to the server. These could include SQL, FTP, industrial protocols, and protocols associated with the Internet of Things.

Global Menus

This is prelimminary documentation and subject to change

The CorsairHMI program can be used on a large number of computers distributed across the Internet. It includes a tree-style menu system that can be used to simplify browser access to the Corsair data that is on these computers. The developer must initialize menu objects with some address information and then menus can grow automatically as new objects are added to the system.

A menu consists of one or more menu objects. Each menu object appears as an item on the menu tree. If an object has entries under it, it is 'parent' to those objects. Those objects are its 'children'. Two objects at the same menu level are 'siblings'.

Menu objects can be of four different types. The first type is a 'Folder'. It is used as a parent to child objects under it. The second type of object is a 'machine link'. It is used to refer to a Corsair computer that has a static public IP address on the Internet. The third type of object is a 'Model Link'. It is used to refer to a Corsair model that is running on a computer. The fourth type of object is a 'Web Link'. It is used to provide an access point to browse to an address on the Internet.

Tennessee Baked Chicken

Assume that you are a lead engineer for Tennessee Baked Chicken. Your company has a chain of baked chicken restaurants all around the world. Your team has designed a leading edge chicken baking machine. It comes in different models with different capabilities. You require extensive remote monitoring and diagnostic capabilities for your chicken` bakers. For yourself, you want a single point of web access that can get you to any Tennessee chicken baker in the world.

Your team has evaluated software alternatives and they have wisely chosen CorsairHMI as the interface to be used on their machines. Each baker has a small Linux-based touchscreen. You've decided that it can go directly to the Internet for a small restaurant with only one machine. For a larger restaurant you want to concentrate the interface from several bakers into a Windows machine that is connected to the Internet.

Your first pilot installation is a single-baker restaurant in Piedmont Missouri. You tie the baker to a wireless router that is hooked to a DSL line. The public static IP is 1.1.1.1. The baking machine's static IP is 192.168.1.73. You set up the router properly, realizing what port forwarding has to be done. You exclude the .73 address from the routers auto-assignment address range.

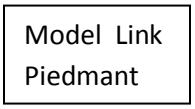
Internet 1.1.1.1 - - - - Router - - - - - - Baker at 192.168.73

Managers IPad with Auto-IP

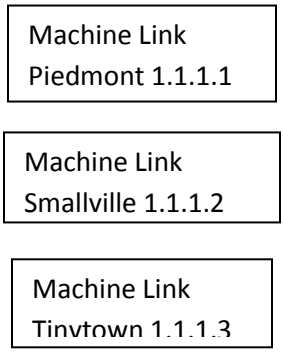
Now the Piedmont restaurant manager can see his baker from his office using the restaurant Wi-Fi and his IPad. When he is at work he browses 192.168.1.73. When he uses his laptop at home he browses 1.1.1.1. For this pilot test your corporate staff also browses 1.1.1.1. Everybody uses a browser shortcut so nobody has to type in an IP every time that they want to look in.

The successful Piedmont installation was followed by single-baker installations at Smallville (1.1.1.2) and Tinytown (1.1.1.3). At this point you figured out that using browser shortcuts to all these locations is not going to work well. You need to implement the menu system.

The first step is to go back to Piedmont, Smallville, and Tinytown and implement a single-object menu at each one. The object is a Model Link object with the name of the restaurant in it. You check the 'Pass Data Upstream' option on the menu system.



The next step is to install a CorsairHMI computer at Tennessee Baked Chicken headquarters. It has no Models of its own, just a menu listing the three restaurants. Each of these objects is a Machine Link with an associated IP. On each of these individual objects you enable the 'Collect Downstream Data' option.

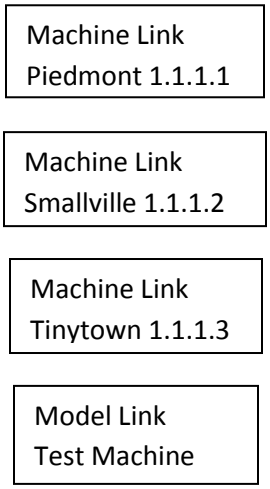


Now if you click on any of the 3 menu options from your headquarters computer Corsair starts your browser pointing to that chicken baking machine.

When you start up the system you notice that the link that you labeled correctly as 'Piedmont' came up as 'Piedmant'. You realize that while you were there doing startup at the restaurant you misspelled the name. Because you set up the menu object to Collect Downstream Data the improper spelling showed up on your machine. You use some remote-control software to correct the spelling on the restaurant machine and your headquarters menu fixes itself automatically.

The next step is to use your headquarters computer to serve out data to any browser over the internet. It goes out as public IP 1.1.1.101. Now any of your staff can browse to that IP and get a menu allowing them to link to any installation.

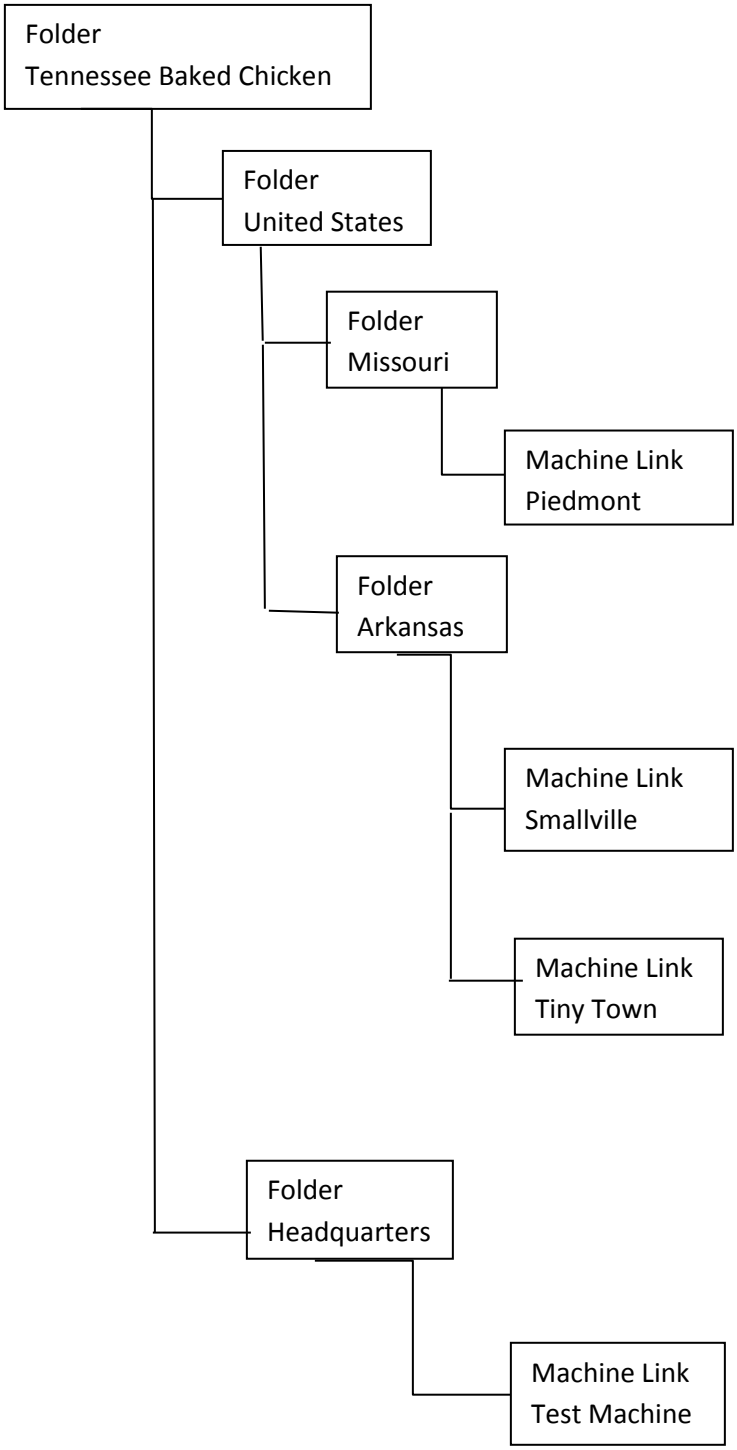
At this point you have a prototype chicken baker at headquarters that you are using as a test machine. You want to be able to monitor it from home. Computers cost money so you decide to install the model on your host machine. You add a Model Link object to your menu.



When you are at the office you can click on any of the restaurant menu objects and browse them. If you click on the Test Machine link you get a local Corsair display of it. From home you can browse to any of the 4 machines.

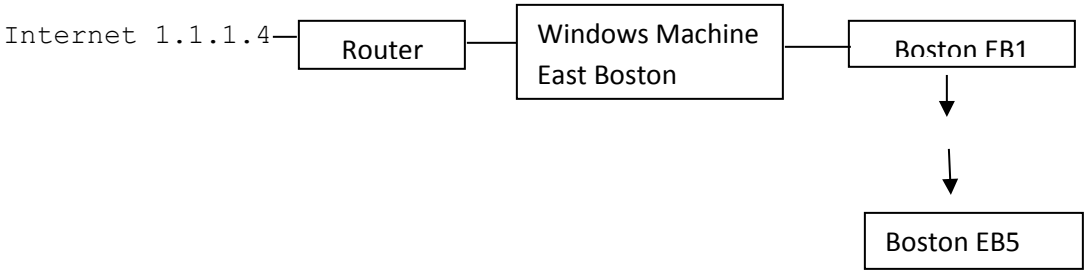
You now understand that the Machine Link menu objects point to computers with Public IP addresses on the Internet. The Model Link menu object points to a Corsair Model on a computer. This is an important step in understanding the Corsair menu system.

Since your boss is gaining interest in this menu system you decide to add some sense of scalability to it. You add some Folder objects.



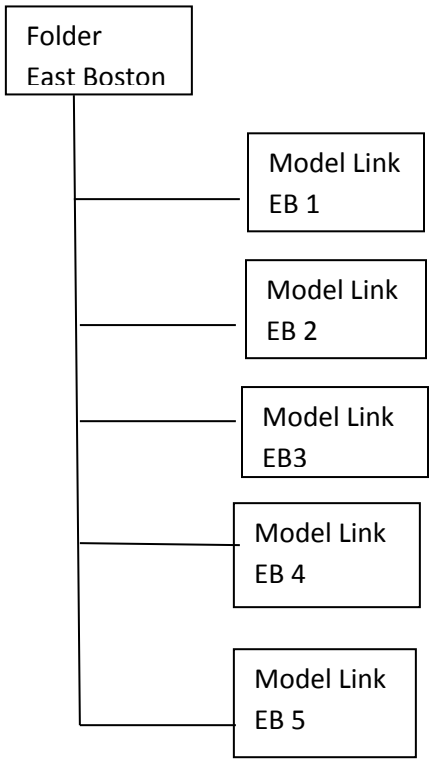
Folder objects do not browse to anything when you click on them. They do not correspond to Corsair computers. They are used to organize menu entries. Now your boss is beginning to understand how big this thing can become.

The next step is a 5-baker installation in a large restaurant in the eastern part of Boston Massachusetts.



Each baker has a small Linux machine just like at the other restaurants. There is a Windows machine with two Ethernet ports. One goes to the 5 chicken bakers, the other goes to the Internet with a public IP of 1.1.1.4. The Windows machine is to act as a data collector for the Linux machines. The individual Linux machines do not have any menu entries. Each of them has one model database corresponding to the type of the machine. They are all set up to be CorsairHMI MBHR hosts.

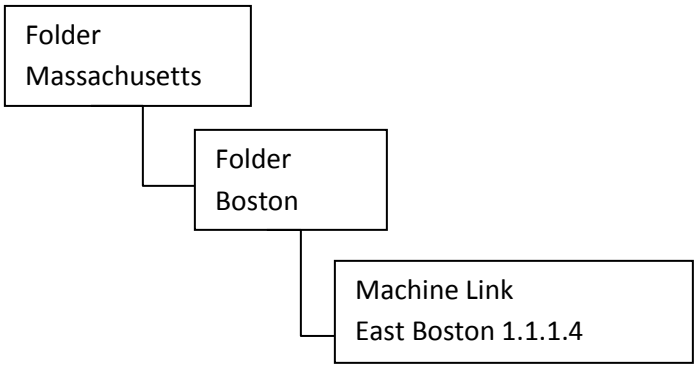
The East Boston Windows machine communicates to each of the bakers using the CorsairHMI extended Modbus protocol. It has copies of each of the 5 models, one from each machine. It has this menu structure:



The ‘Pass Data Upstream’ option is enabled on the Windows machine.

If the East Boston manager is sitting at the Windows machine he can click on menu objects for any of his 5 bakers and monitor it at Corsair extended Modbus speeds. He can go to any of the bakers and use its local Linux display. He can browse from home into 1.1.1.4.

Since you want to get in on the action at East Boston you go back to headquarters and add the following to your menu:



You turn on the ‘Collect Downstream Data’ option on the East Boston Machine Line. When you come in the next day you discover that all 5 of the East Boston machines have been added automatically as Model Links on your menu. If East Boston adds or deletes a machine locally your menu will be updated to reflect the change.

Clicking on the ‘East Boston’ Folder brings a browser to 1.1.14 and the East Boston menu which includes the 5 machines. Clicking on any of the 5 machines brings a browser to 1.1.1.4 with the correct initial web page for that machine.

Now a few years later the system is completely implemented worldwide. Your favorite data center has rented you servers labeled:

Tennessee Baked Global

United States

Canada

Europe

Asia

Australia

Africa

Each of these servers has a menu with links to other machines. Some of these machines are directly on chicken bakers and some are collector machines that are hooked to multiple bakers. The country machines are linked to the global machine. Any time that a restaurant is added its IP address must be added as a Machine Link menu object on the immediately upstream Corsair computer. The menu system will pass the menu information upstream all the way to the Global computer. Every chicken baker in the world will appear on that menu tree. Users can browse into any machine and access anything that is under it in the menu structure.

There can be a few hundred levels in a Corsair menu. Menu options can correspond to physical machines or to organizational menu links at the developer's discretion. Model links are limited to 100 per instance of the Corsair software. Total menu options can be in the tens of thousands.

PLC Clock Logic

Siemens S7 PLCs

This code was tested on a Siemens S7-1200. It should work on an S7-1500. It will not work on an S7-400.

The PLC Clock Data block is a 15-register long table of unsigned integers.

This is tagged in the PLC as Clock_Data_0 to Clock_Data_14 with type UInt.

There is a Cyclic Interrupt routine named Clock_Handling.

It repeats at a 250 millisecond interval.

It uses the SCL language.

Clock_Read_Data and Clock_Write_Data are local to the routine with type DTL.

Clock_Read_Retval and Clock_Write_Retval are local to the routine with type Int.

Power_Up is a global bit that comes on 30 seconds after the first scan.

This is the SCL code:

```
#Clock_Read_Retval := RD_LOC_T(#Clock_Read_Data);
IF (#Clock_Read_Retval = 0) OR (#Clock_Read_Retval = 1) THEN
   "Clock_Data_0" := #Clock_Read_Data.SECOND;
   "Clock_Data_1" := #Clock_Read_Data.MINUTE;
   "Clock_Data_2" := #Clock_Read_Data.HOUR;
   "Clock_Data_3" := #Clock_Read_Data.WEEKDAY;
   "Clock_Data_4" := #Clock_Read_Data.DAY;
   "Clock_Data_5" := #Clock_Read_Data.MONTH;
   "Clock_Data_6" := #Clock_Read_Data.YEAR;
   IF "Power_Up" AND ("Clock_Data_14" <> 0) THEN
      #Clock_Write_Data := #Clock_Read_Data;
      IF ("Clock_Data_14" AND 1) <> 0 THEN
         #Clock_Write_Data.SECOND := "Clock_Data_7";
      END_IF ;
      IF ("Clock_Data_14" AND 2) <> 0 THEN
         #Clock_Write_Data.MINUTE := "Clock_Data_8";
      END_IF ;
```

```
      IF ("Clock_Data_14" AND 4) <> 0 THEN
         #Clock_Write_Data.HOUR := "Clock_Data_9";
      END_IF ;
      IF ("Clock_Data_14" AND 16) <> 0 THEN
         #Clock_Write_Data.DAY := "Clock_Data_11";
      END_IF ;
      IF ("Clock_Data_14" AND 32) <> 0 THEN
         #Clock_Write_Data.MONTH := "Clock_Data_12";
      END_IF ;
      IF ("Clock_Data_14" AND 64) <> 0 THEN
         #Clock_Write_Data.YEAR := "Clock_Data_13";
      END_IF ;
      #Clock_Write_Retval := WR_LOC_T(LOCTIME:= #Clock_Write_Data, DST:= 0);
   END_IF ;
END_IF;
"Clock_Data_14" := 0;  (* Reset Control Bits *)
```


Clock Logic is especially complex in a Quantum Processor because of the packing of the data into bytes of registers using a Binary Coded Decimal (BCD) data format. Corsair expects clock data to be in a commonly used binary format. The code must take into account the Quantum clock control bits and operate them in the proper sequence.

This sample code uses an integer array named ‘Clock_Data’ with a size of 15. It’s elements are Clock_Data[1] to Clock_Data[15]. It is a located variable with a %MW address that can be read and written by the Corsair interface. ‘Power_Up’ is a memory tag that has stays false for a short period of time when the CPU starts to run. This prevents the CPU from acting on clock set data that arrives while it is halted. The end user will have to write logic for this tag.

The Day of Week setting logic may not be needed if the Quantum calculates day of week from the rest of the date.

```
(* See if the clock data is good *)
IF %S51 = 0 THEN
  (* Clock data is good *)
  (* Get seconds *)
  WORD_AS_BYTE(INT_TO_WORD(%SW50), Low_Byte, High_Byte) ;
  Clock_Data[1] := BCD_TO_INT(BYTE_TO_INT(High_Byte)) ;
  (* Get minutes *)
  WORD_AS_BYTE(INT_TO_WORD(%SW51), Low_Byte, High_Byte) ;
  Clock_Data[2] := BCD_TO_INT(BYTE_TO_INT(Low_Byte)) ;
  (* Get hour *)
  Clock_Data[3] := BCD_TO_INT(BYTE_TO_INT(High_Byte)) ;
  (* Get Weekday *)
  WORD_AS_BYTE(INT_TO_WORD(%SW49), Low_Byte, High_Byte) ;
  Int_Temp := BCD_TO_INT(BYTE_TO_INT(Low_Byte)) + 1 ;
  IF Int_Temp = 8 THEN
    Int_Temp := 1 ;
```

```
  END_IF ;
  Clock_Data[4] := Int_Temp ;
  (* Get Day of Month *)
  WORD_AS_BYTE(INT_TO_WORD(%SW52), Low_Byte, High_Byte) ;
  Clock_Data[5] := BCD_TO_INT(BYTE_TO_INT(Low_Byte)) ;
  (* Get Month *)
  Clock_Data[6] := BCD_TO_INT(BYTE_TO_INT(High_Byte)) ;
  (* Get the year *)
  Clock_Data[7] := BCD_TO_INT(%SW53) ;
  (* Copy to nicely labeled variables for the PLC to use *)
  Clock_Month := Clock_Data[6] ;
  Clock_Day := Clock_Data[5] ;
  Clock_Hour := Clock_Data[3] ;
  Clock_Minute := Clock_Data[2] ;
END_IF ;

%S50 := FALSE ;
(* Clock Setting *)
IF ((Power_Up <> 0) AND (Clock_Data[15] <> 0)) THEN
  (* Turn on system bit to freeze registers *)
  %S50 := TRUE ;
  (* Set Second *)
  IF((INT_TO_WORD(Clock_Data[15]) AND 16#0001) <> 0) THEN
    WORD_AS_BYTE(INT_TO_WORD(%SW50), Low_Byte, High_Byte) ;
    High_Byte := INT_TO_BYTE(INT_TO_BCD(Clock_Data[8])) ;
```

```
  %SW50 := WORD_TO_INT(BYTE_AS_WORD(Low_Byte, High_Byte));
END_IF ;
(* Set Minute *)
IF((INT_TO_WORD(Clock_Data[15]) AND 16#0002) <> 0) THEN
  WORD_AS_BYTE(INT_TO_WORD(%SW51), Low_Byte, High_Byte) ;
  Low_Byte := INT_TO_BYTE(INT_TO_BCD(Clock_Data[9])) ;
  %SW51 := WORD_TO_INT(BYTE_AS_WORD(Low_Byte, High_Byte));
END_IF ;
(* Set Hour *)
IF((INT_TO_WORD(Clock_Data[15]) AND 16#0004) <> 0) THEN
  WORD_AS_BYTE(INT_TO_WORD(%SW51), Low_Byte, High_Byte) ;
  High_Byte := INT_TO_BYTE(INT_TO_BCD(Clock_Data[10])) ;
  %SW51 := WORD_TO_INT(BYTE_AS_WORD(Low_Byte, High_Byte));
END_IF ;
(* Set Weekday *)
IF((INT_TO_WORD(Clock_Data[15]) AND 16#0008) <> 0) THEN
  Int_Temp := Clock_Data[11] - 1;
  IF Int_Temp < 1 THEN
    Int_Temp := 7 ;
  END_IF ;
  %SW49 := INT_TO_BCD(Int_Temp) ;
END_IF ;
(* Set Day of Month *)
IF((INT_TO_WORD(Clock_Data[15]) AND 16#0010) <> 0) THEN
  WORD_AS_BYTE(INT_TO_WORD(%SW52), Low_Byte, High_Byte) ;
```

```
  Low_Byte := INT_TO_BYTE(INT_TO_BCD(Clock_Data[12])) ;
  %SW52 := WORD_TO_INT(BYTE_AS_WORD(Low_Byte, High_Byte));
 END_IF ;
 (* Set Month *)
 IF((INT_TO_WORD(Clock_Data[15]) AND 16#0020) <> 0) THEN
  WORD_AS_BYTE(INT_TO_WORD(%SW52), Low_Byte, High_Byte) ;
  High_Byte := INT_TO_BYTE(INT_TO_BCD(Clock_Data[13])) ;
  %SW52 := WORD_TO_INT(BYTE_AS_WORD(Low_Byte, High_Byte));
 END_IF ;
 (* Set Year *)
 IF((INT_TO_WORD(Clock_Data[15]) AND 16#0040) <> 0) THEN
  %SW53 := INT_TO_BCD(Clock_Data[14]) ;
 END_IF ;
END_IF ;
Clock_Data[15] := 0;
```

General Electric 90/30

%R2037 Time Set Temp

%R2038 Time Calc Temp

%R2039 Binary Month and Year

%R2040 Binary Hours and DOM

%R2041 Binary Seconds and Minutes

%R2042 Binary Day of the Week

%R2043 Clock Service 0-Read 1-Set

%R2044 Clock Format 1-BCD

%R2045 Clock Month and Year

%R2046 Clock Hours and DOM

%R2047 Clock Seconds and Minutes

%R2048 Clock Day of Week

%R3001 Clock_Block_1 Read Second

%R3002 Clock_Block_2 Read Minute

%R3003 Clock_Block_3 Read Hour

%R3004 Clock_Block_4 Read Day of Week

%R3005 Clock_Block_5 Read Day of Month

%R3006 Clock_Block_6 Read Month

%R3007 Clock_Block_7 Read Year

%R3008 Clock_Block_8 Set Second

%R3009 Clock_Block_9 Set Minute

%R3010 Clock_Block_10 Set Hour

%R3011 Clock_Block_11 Set Day of Week

%R3012 Clock_Block_12 Set Day of Month

%R3013 Clock_Block_13 Set Month

%R3014 Clock_Block_14 Set Year

%R3015 Clock_Block_15 Clock Control Bits

%T251 Flag_1 Indicates that the read is successful

%T252 Flag_2 Indicates that the control bits are nonzero

The code goes into a block typically named ‘Clock’.

It is called at 0.25 second intervals.

Rung #1

MOVE INT Length 1 IN 0 Q %R2043 Sets Service to Read

MOVE INT Length 1 IN 1 Q %R2044 Sets Format to BCD

SVC_REQ FNC 7 PRM %R2043 Top feeds Coil %T251

This rung reads the clock.

It turns on %T251 on success.

Rung #2

Normally Open Contact %T251

BCD4 TO INT IN %R2045 Q %R2039

BCD4 TO INT IN %R2046 Q %R2040

This rung converts the Month and Year to Binary.

It also converts the Hours and DOM to Binary.

Rung #3

Normally Open Contact %T251

BCD4 TO INT IN %R2047 Q %R2041

BCD4 TO INT IN %R2048 Q %R2042

This rung converts the Seconds and Minutes to Binary.

It also converts the Day of Week to Binary.

Rung #4

Normally Open Contact %T251

DIV INT IN1 %R2039 IN2 100 Q %R3006

MOD INT IN1 %R2039 IN2 100 Q %R2038

This rung divides by 100 to get the Month for the interface.

The MOD operation is to get the last two digits of the year.

It goes into the %R2038 temporary.

Rung #5

Normally Open Contact %T251

ADD INT IN1 %R2038 IN2 2000 Q %R3007

This rung adds 2000 to the %R2038 temporary.

The result is the current year for the interface.

Rung #6

Normally Open Contact %T251

MOD INT IN1 %R2040 IN2 100 Q %R3005

MON INT IN1 %R2042 IN2 100 Q %R3004

This rung sends the Day of Month and Day of Week to the interface.

Rung #7

Normally Open Contact %T251

DIV INT IN1 %R2040 IN2 100 Q %R3003

MOD INT IN1 %R2041 IN2 100 Q %R3002

DIV INT IN1 %R2041 IN2 100 Q %R3001

The first DIV sends the Hours to the interface.

The MOD sends the Minutes to the interface.

The second DIV sends the Seconds to the interface.

Rung #8

No contact

NE INT IN1 %R3015 IN2 0 Q to coil %T252

This rung turns on %T252 if the %R3015 clock control bits are nonzero.

Rung #9

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 1 Q feeds the MUL Block

MUL INT IN1 %R3008 IN2 100 Q %R2037 Top feeds the INT TO BCD4

INT TO BCD4 IN %R2037 Q %R2037 Top feeds a continuation Plus coil

Rung #10

Plus Continuation Contact

AND WORD IN1 %R2047 IN1 255 Q %R2047 Top feeds the OR WORD

OR WORD IN1 %R2047 IN2 %R2037 Q %R2047

Rungs 9 and 10 handle the setting of seconds.

Rung #11

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 2 Q feeds the INT TO BCD4

INT TO BCD4 IN %R3009 Q %R2037 Top feeds the AND WORD

AND WORD IN1 %R2047 IN2 65280 Q %R2047 Top feeds a continuation plus

Rung #12

Plus Continuation Contact

OR WORD IN1 %R2047 IN2 %R2037 Q %R2047

Rungs 11 and 12 handle the setting of minutes.

Rung #13

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 3 Q Feeds the MUL INT

MUL INT IN1 %R3010 IN2 100 Q %R2037 Top feeds the INT TO BCD4

INT TO BCD4 IN %R2037 Q %R2037 Top feeds a continuation plus coil

Rung #14

Plus Continuation Contact

AND WORD IN1 %R2048 IN2 255 Q %R2048 Top feeds the OR WORD

OR WORD IN1 %R2046 IN2 %R2037 Q %R2046

Rungs 13 and 14 handle the settings of hours.

Rung #15

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 5 Q feeds the INT TO BCD4

INT TO BCD4 IN %R3012 Q %R2037 Top feeds the AND WORD

AND WORD IN1 %R2048 IN2 65280 Q %R2046 Top feeds a continuation plus

Rung #16

Plus Continuation Contact

OR WORD IN1 %R2046 IN2 %R2037 Q %R2046

Rungs 15 and 16 handle the setting of Day of Month.

Rung #17

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 6 Q feeds the MUL INT

MUL INT IN1 %R3013 IN2 100 Q %R2037 Top feeds the INT TO BCD4

INT TO BCD4 IN %R2037 Q %R2037 Top feeds a continuation plus coil

Rung #18

Plus Continuation Contact

AND WORD IN1 %R2045 IN2 255 Q %R2045 Top feeds the OR WORD

OR WORD IN1 %R2045 IN2 %R2037 Q %R2045

Rungs 17 and 18 handle the setting of Month.

Rung #19

No Contact

BIT TEST WORD Length 1 IN %R3015 BIT 7 Q feeds the MOD INT

MOD INT IN1 %R3014 IN2 100 Q %R2037 Top feeds the AND WORD

AND WORD IN1 %R2045 IN2 65280 Q %R2045 Top feeds a continuation plus

Rung #20

Plus Continuation Contact

INT TO BCD4 IN %R2037 Q %R2037 Top feeds the OR WORD

OR WORD IN1 %R2045 IN2 %R2037 Q %R2045

Rungs 19 and 20 handle the setting of Year.

Rung #21

Normally Open Contact %T252

MOVE INT Length 1 IN 1 Q %R2043 Sets Service to Set

MOVE INT Length 1 IN 1 Q %R2044 Sets Format to BCD

SVC REQ FNC 7 PRM %R2043

This rung does the clock set operation.

Rung #22

No Contact

BLK CLR WORD Length 1 IN %R3015

This rung resets the clock control bits to zero.

Allen-Bradley ControlLogix

This Ladder Routine needs to be executed at 0.25 second intervals. It has a local program tag named 'Time_Data' which is a DINT array with a size of 7. It is used for the Get System Value and Set System Value functions. It's elements are:

[0] Year

[1] Month 1-12

[2] Day 1-31

[3] Hour 0-23

[4] Minute 0-59

[5] Seconds 0-59

[6] Microseconds

Note the unforgivable omission of Day of Week in the AB instruction. It is usually more important than the year.

There is a Global tag named 'Clock_Data'. It is an INT array with a size of 15 that is read and written by the Corsair interface.

Rung #0

GSV Get System Value

Class Name	WallClockTime
Instance Name	(Blank)
Attribute Name	LocalDateTime
Source	Time_Data[0]

The PLC clock read command.

Rung #1

No Contact

MOV Move

Source	Time_Data[5]
Dest	Clock_Data[0]

Sends Seconds to the interface.

Rung #2

No Contact

MOV Move

Source	Time_Data[4]
Dest	Clock_Data[1]

Sends Minutes to the interface.

Rung #3

No Contact

MOV Move

Source	Time_Data[3]
Dest	Clock_Data[2]

Sends Hours to the interface.

Rung #4

No Contact

MOV Move

Source	0
Dest	Clock_Data[3]

Zeros Day of Week for the interface.

Rung #5

No Contact

MOV Move

Source Time_Data[2]

Dest Clock_Data[4]

Sends Day of Month to the interface.

Rung #6

No Contact

MOV Move

Source Time_Data[1]

Dest Clock_Data[5]

Sends Month to the interface.

Rung #7

No Contact

MOV Move

Source Time_Data[0]

Dest Clock_Data[6]

Sends Year to the interface.

Rung #8

Normally Open Contact Clock_Data[14].0

MOV Move

Source Clock_Data[7]

Dest Time_Data[5]

Sets the Seconds.

Rung #9

Normally Open Contact Clock_Data[14].1

MOV Move

Source Clock_Data[8]

Dest Time_Data[4]

Sets the Minutes.

Rung #10

Normally Open Contact Clock_Data[14].2

MOV Move

Source Clock_Data[9]

Dest Time_Data[3]

Sets the Hour.

Rung #11

Normally Open Contact Clock_Data[14].4

MOV Move

Source Clock_Data[11]

Dest Time_Data[2]

Sets the Day of Month.

Rung #12

Normally Open Contact Clock_Data[14].5

MOV Move

Source Clock_Data[12]

Dest Time_Data[1]

Sets the Month.

Rung #13

Normally Open Contact Clock_Data[14].6

MOV Move

Source Clock_Data[13]

Dest Time_Data[0]

Sets the Year.

Rung #14

NEQ Source A Clock_Data[14] Source B 0

SSV Set System Value

Class Name WallClockTime

Instance Name (Blank)

Attribute Name LocalDateTime

Source Time_Data[0]

The PLC clock write command.

Rung #15

No Contact

CLR Clear

Dest Clock_Data[14]

This rung clears the clock control bits.

Bootp Equipment

Allen-Bradley 2100-ENT Ethernet to DeviceNet Converter

You will want to read the Ethernet MAC address from the side of the module if possible. If there is an existing IP address that you know go to procedure ‘A’. If there is no IP address like with a new module go to procedure ‘B’.

Allen-Bradley SMC-Flex Soft Start

If the starter has an integrated HMI use it to set the IP. If there is an existing IP address that you know go to procedure ‘A’. If there is no IP address like with a new module go to procedure ‘B’.

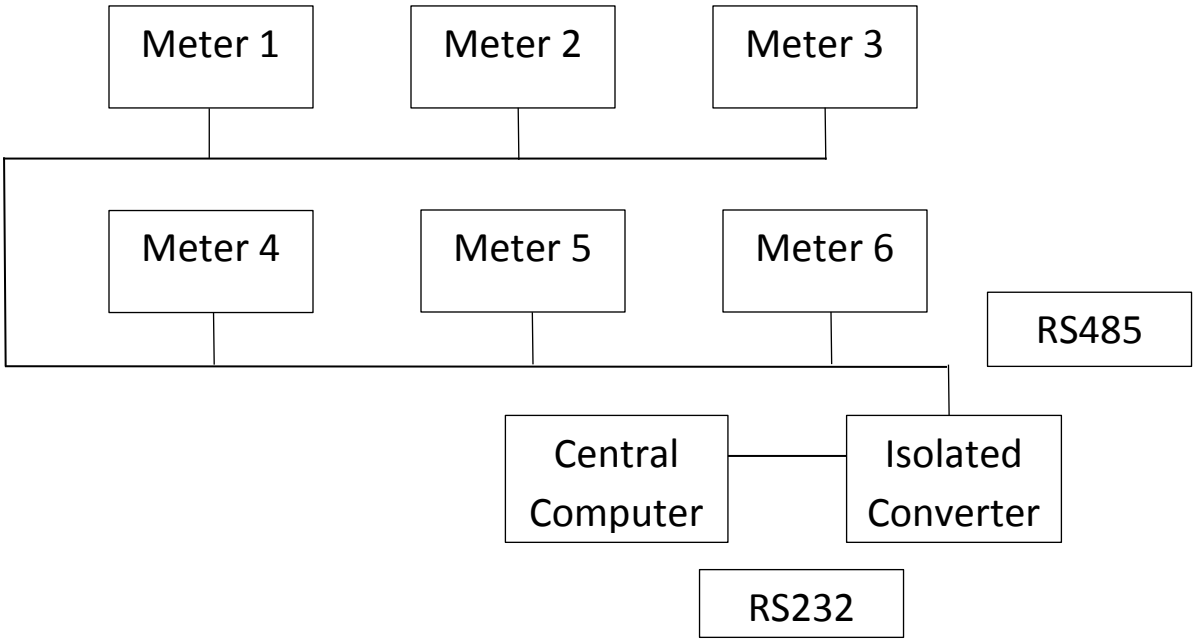
Modicon M1 Momentum Ethernet CPU

This applies only to the CPU modules. The Ethernet Remote I/O top-hat has not been tested. The Transmit-Only BootP option on Corsair has been designed to emulate Modicon’s BootpLT software. If there is an existing IP address that you know go to procedure ‘C’.

If you do not know the existing address go to procedure ‘D’. After the new IP address is loaded with Corsair do not cycle the power to the PLC. The PLC programming software must be used to load a configuration to the CPUs flash memory before power is removed.

Campus Power Monitoring

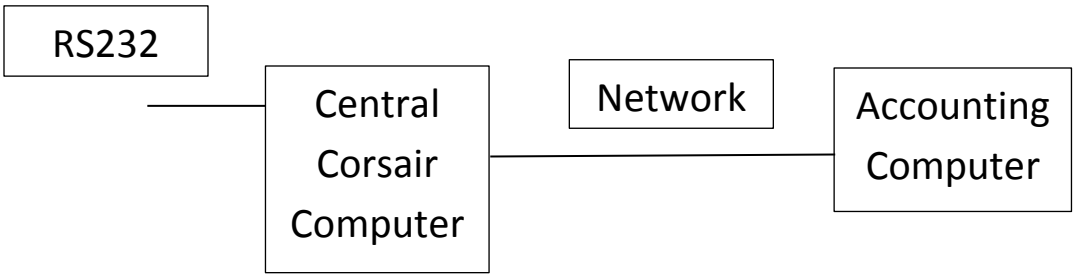
Charlie is a maintenance man at a major university. The campus consists of three clusters of buildings: 8 in the north, 6 in the center, and 11 in the south. Charlie was given orders to monitor and log power usage on each of the buildings in the central cluster. He found metering equipment that could be hooked together using 2-wire RS-485 Modbus RTU communications. He purchased 6 meters and a computer.



Charlie needed a data converter to match the RS-232 port on his computer to the 2-wire RS-485 wiring required by the meters. He spent a little extra money to get an isolated converter that would keep electrical noise from the meters out of his computer. Each meter had to be set up with a unique Modbus ID number. A meter fresh from the factory used ID one. Charlie used ID numbers two through seven for his six meters so if anyone hooked up an additional meter or if a meter was sent out for repair he would have less troubles with address conflicts. He used the meter's default 9600 baud data rate with no parity. The Modbus RTU protocol requires 8 data bits.

The next step was to purchase Corsair software for the computer. The Corsair dealer helped Charlie to develop a data template that matched the Modbus register map of the meter. He used the Corsair Modbus RTU serial driver and began reading values.

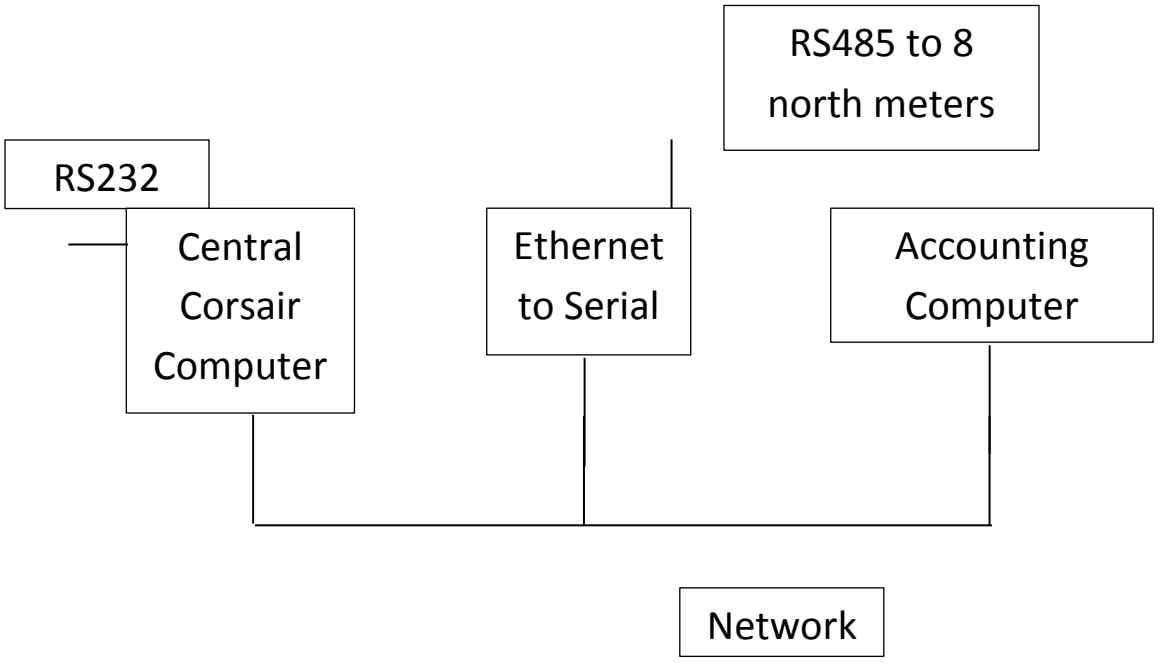
The next week Charlie was told that Linda in accounting needed to see information from the power meters on her computer. She needed read-only access since she wouldn't be changing any data. Her computer was already on the campus Ethernet network so Charlie added the Corsair computer to that network.



Charlie bought some remote-control software for the Corsair computer. Linda could now take control of the computer any time that she wanted to from her office. This worked, but it was clumsy. It seemed that she always wanted to take control at the same time that Charlie wanted to see something. He had to work through a few issues with setting up printer drivers. The real problem hit when the guys in the campus MIS department found out that Charlie had installed the remote control software without checking with them. Their security policies prohibited any use of this type of program. Charlie had to find another solution. He called his Corsair dealer.

After a little study, Charlie configured his Corsair computer to act as an MB host. He bought an inexpensive MB remote-only version of the Corsair software to run on the accounting computer. Now she could see everything that Charlie could see on the central computer anytime that she wanted to.

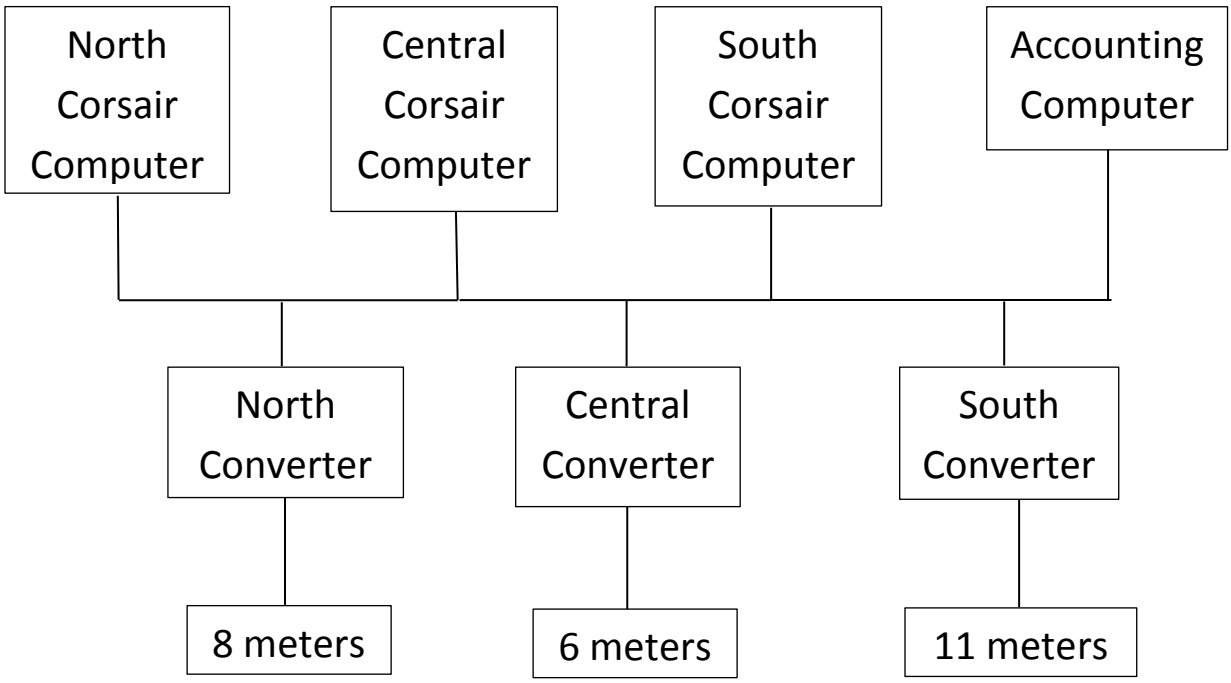
The next request was to put meters on the 8 north campus and 11 south campus buildings. Charlie did the north first. He purchased an inexpensive Modbus serial to Ethernet converter.



Charlie set up a second driver in his Corsair database to use the Modbus over Ethernet (Modbus Application) protocol. He had to contact Joe from MIS to get an IP address for the converter. Joe reluctantly assigned it. The data communications worked as expected. Charlie updated the Corsair database on the accounting computer to include the north campus buildings.

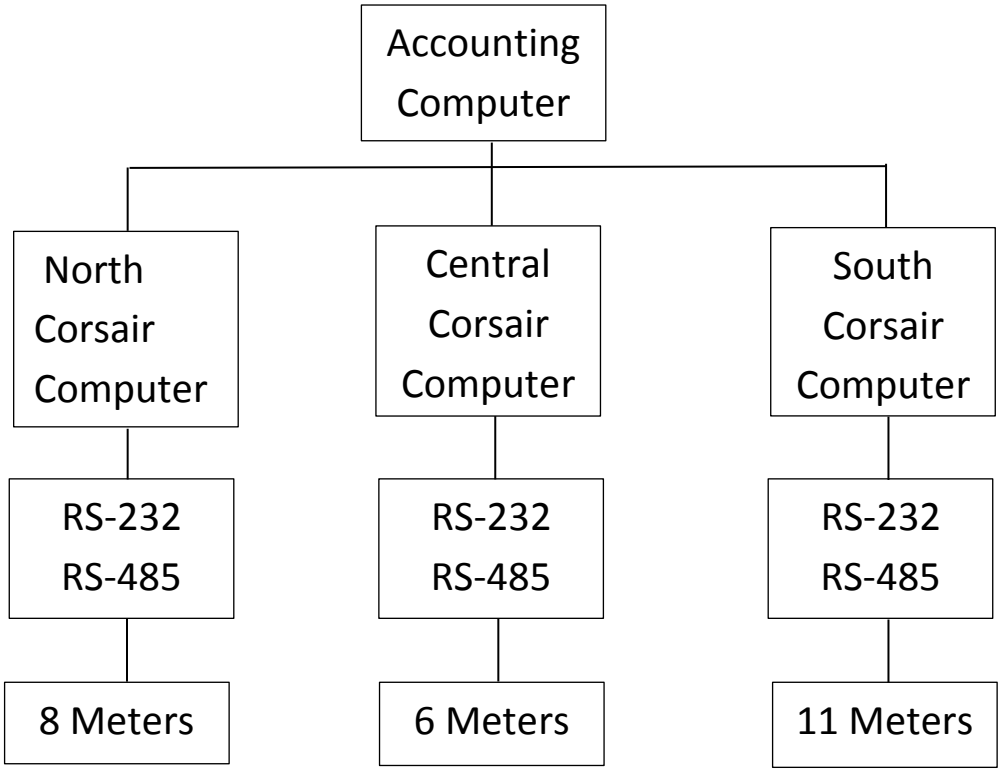
Next Charlie went to work installing the 11 meters on the south campus. He hooked them to another serial to Ethernet converter. MIS gave him another IP address. When it was powered up Charlie couldn't get good communications even though he did exactly what he had done on the north. He called his Corsair dealer and they reviewed the system architecture. At the dealer's suggestion Charlie checked how many TCP/IP connections his converters could support at one time. The answer was 10. He was about to buy another converter to split up the meters when he talked to the distributor again. The distributor told Charlie how the Corsair single-socket Ethernet multi-drop driver works. This driver is specially written to use with this type of converter. Charlie changed the driver type in the Corsair database and the south campus worked.

Everybody was happy until the holiday weekend then the central Corsair computer failed and they lost all the logging data for three days. Charlie was told that each building needed to log all of its' data and also all of the data from the other buildings. He purchased two more computers and another serial to Ethernet converter.



This approach turned out to be a disaster. Each serial to Ethernet converter now had to keep 3 computers happy instead of 1. The 9600 baud serial data rate was too slow to get a usable refresh rate on the Corsair computers. Charlie began to panic. He called his Corsair dealer again.

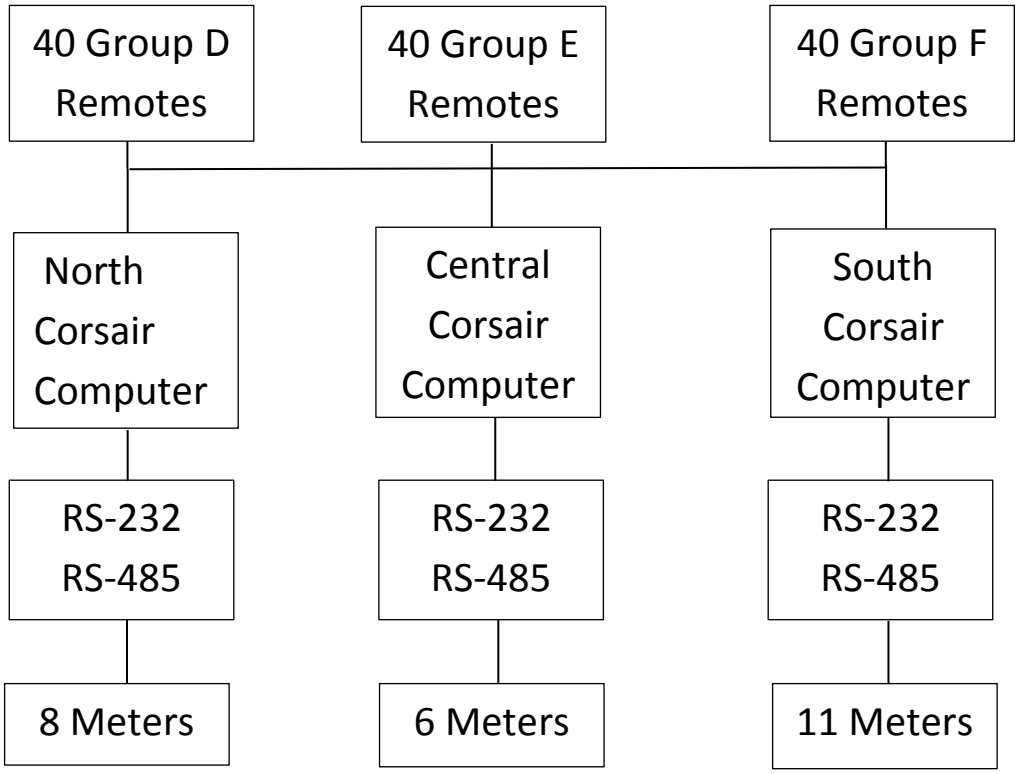
The dealer suggested that Charlie find his old RS-232 to 485 isolated converter and buy two more like it.



Each Corsair computer was configured as an MB host. A little switching around of drivers and database addresses allowed each computer to read its meters over the serial line and the other meters over the network. The speed was restored and data was logged for all three buildings on all three computers. MIS was happy to not have strange converters on their network. Accounting still used the central Corsair computer as its primary host. Charlie added the north computer as its backup host. He figured that the project was finally done. It was for about two weeks.

The university management team was so proud of their power monitoring system that they wanted everyone on the network to be able to view the meter data. This totaled 120 computers, including the one in accounting. Charlie contacted his Corsair dealer again and asked for volume pricing on the MB remote-only version of the Corsair software.

Charlie developed his Corsair database in several steps as his system evolved. This document details a possible way to do it starting from an empty database.



The system consists of a total of 123 Corsair computers. The North, Central, and South computers run a full featured version of the Corsair software. Each is hooked to a group of meters through a serial line. 120 of them run less expensive MB Remote-only versions of the software. They are divided into 3 groups

of 40. These groups are labeled group D, group E, and group F. For this example, the remote computers are named D1 through D40, E1 through E40 and F1 through F40.

The remote computer database is developed first. It must then be slightly modified for each of the 3 host computers. There will be a total of 4 different Corsair .CAP files. All computers will be able to see the data from all the meters.

The first step is to determine IP addresses for the North, Central and South host computers. They must use fixed IP addresses that do not change. Assume that they are to be 192.168.141.1 through 3 for this example. There addresses must be assigned to the network adaptor under the Windows Control Panel network configuration. They also must be entered as MIS addresses onto computer records in the Corsair database. 120 remote computers may have server-assigned IP addresses. Six records must be entered into the Corsair computer database:

Record 1:
Computer Name : North Campus
MIS IP Address: 192.168.141.1
MB Host?: Yes
MB Remote?: No
Can Write? : No
Primary Host: ??
Backup Host: ??
Extended Protocol: No

Record 2:
Computer Name: Central Campus
MIS IP Address: 192.168.141.2
MB Host?: Yes
MB Remote?: No
Can Write?: No
Primary Host: ??
Backup Host: ??
Extended Protocol: No

Record 3:
Computer Name: South Campus
MIS IP Address: 192.168.141.3
MB Host?: Yes
MB Remote?: No
Can Write?: No
Primary Host: ??
Backup Host: ??
Extended Protocol: No

Record 4
Computer Name: Group D Remote

MIS IP Address: 0.0.0.0
MB Host?: No
MB Remote?: Yes
Can Write?: No
Primary Host: North Campus
Secondary Host: Central Campus
Extended Protocol: Yes

Record 5
Computer Name: Group F Remote
MIS IP Address: 0.0.0.0
MB Host?: No
MB Remote?: Yes
Can Write?: No
Primary Host: Central Campus
Secondary Host: South Campus
Extended Protocol: Yes

Record 6
Computer Name: Group I Remote
MIS IP Address: 0.0.0.0
MB Host?: No
MB Remote?: Yes
Can Write?: No
Primary Host: South Campus
Secondary Host: North Campus
Extended Protocol: Yes

This database arrangement distributes the 120 remotes evenly across the 3 hosts. (40 each) A host can support up to 100 remotes. If one of the N, C, or S hosts fails each remote will still find a host to connect with.

The next step is to create a single driver record in the Corsair database. It gets the Modbus RTU serial driver type. Zooming on it allows the creation of multiple device tags on the PLC. The addresses and type of these devices depend upon the Modbus register map of the meter. Every device tag that is needed should be created before any cloning is done. It's less work to delete extras later than it is to add them in after cloning is done. When all the device tags are finished the PLC record needs to be cloned 10 times to create a total of 11 PLCs. They get node ID numbers from 2 through 12. Partial renaming on PLC and device tags can occur now.

The next step is to clone the driver record twice for a total of 3 drivers. This will result in a total of 33 PLCs in the database. Extra PLC records need to be deleted from the north and center drivers. Suggested addresses would be:

Driver Record: North Meters

PLC: North Meter 1 Node:2 MBHR ID: 2

PLC:	North Meter 2	Node:3	MBHR ID: 3
PLC:	North Meter 3	Node:4	MBHR ID: 4
PLC:	North Meter 4	Node:5	MBHR ID: 5
PLC:	North Meter 5	Node:6	MBHR ID: 6
PLC:	North Meter 6	Node:7	MBHR ID: 7
PLC:	North Meter 7	Node:8	MBHR ID: 8
PLC:	North Meter 8	Node:9	MBHR ID: 9

Driver Record: Center Meters

PLC:	Center Meter 1	Node:2	MBHR ID: 22
PLC:	Center Meter 2	Node:3	MBHR ID: 23
PLC:	Center Meter 3	Node:4	MBHR ID: 24
PLC:	Center Meter 4	Node:5	MBHR ID: 25
PLC:	Center Meter 5	Node:6	MBHR ID: 26
PLC:	Center Meter 6	Node:7	MBHR ID: 27

Driver Record: South Meters

PLC:	South Meter 1	Node: 2	MBHR ID: 42
PLC:	South Meter 2	Node: 3	MBHR ID: 43
PLC:	South Meter 3	Node: 4	MBHR ID: 44
PLC:	South Meter 4	Node: 5	MBHR ID: 45
PLC:	South Meter 5	Node: 6	MBHR ID: 46
PLC:	South Meter 6	Node: 7	MBHR ID: 47
PLC:	South Meter 7	Node: 8	MBHR ID: 48
PLC:	South Meter 8	Node: 9	MBHR ID: 49

PLC:	South Meter 9	Node:10	MBHR ID: 50
PLC:	South Meter 10	Node:11	MBHR ID: 51
PLC:	South Meter 11	Node:12	MBHR ID: 52

At this point the database contains a large number of duplicate tag errors on device records. It's time to finish renaming all tags to clear up duplication. The Corsair tree printout may assist in checking the database at this point.

The next step is development of all the operator screens and sheets that are required for the project. Each record should be as complete and correct as possible before cloning and retagging.

It is now recommended to print the complete project tree. If each meter is developed identically the usage counts on each device record should match the corresponding counts on other PLCs.

PART 3 CREATING THE DATA LOGS

The campus system requires that log data be kept on each of the North, Central, and South host computers. The 120 remote computers will not be running the Corsair program continuously so they each need to get their log data from a host. The group D remotes will get their log data from the North host, group E from the Central and group F from the South.

PART 4 MODIFYING THE DATABASE FOR EACH OF THE HOST COMPUTERS

At this point the database is ready to be installed on each of the 120 Corsair remote computers. The Corsair.cfg configuration file must be properly set up depending on whether each computer is a group D, E, or F remote. This is done under the computer properties menu selection.

The remotes communicate to the hosts via Ethernet so the serial port settings on the remotes are not important. Port 1 on each of the three hosts should be set to 9600 baud with 8 data bits, one stop bit, and parity as required by the metering equipment.

Each of the 3 driver records in the Corsair database now has the Modbus RTU serial type. Each host can only access one group of meters through a serial port. It must access the other two groups of meters through Ethernet. This is doe by changing some driver types to Modbus Multi-Drop Extended protocol. This is done as follows:

North Computer

Driver #1 - North meters - RTU Serial

Set port to #1

Driver#2 - Central meters - Multi-Drop Extended

Set all PLC IP addresses to 192.168.141.2

Driver#3 - South meters - Multi-Drop Extended

Set all PLC IP addresses to 192.168.141.3

Central Computer

Driver#1 - North meters - Multi-Drop Extended

Set al l PLC IP addresses to 192.168.141.1

Driver#2 - Central meters - RTU Serial

Set Port to #1

Driver#3 - South meters - Multi-Drop Extended

Set all PLC IP addresses to 192.168.141.3

South Computer

Driver#1 - North meters - Multi-Drop Extended

Set all PLC IP addresses to 192.168.141.1

Driver#2 - Central meters - Multi-Drop Extended

Set all PLC addresses to 192.168.141.2

Driver#3 - South meters - RTU Serial

Set port to #1

Each host will appear as an MB remote to the other two hosts. When everything is running each host will show a total of 42 connected remotes – 40 remotes and the two other hosts.

RFIDEas Reader

This is preliminary documentation which is subject to change

This application note describes options for integrating an RFIDeas ID reader with CorsairHMI software using the EtherIP protocol.

The unit used to develop the software was an RDR-800W1AKB-P. It is described as an 'RFIDeas pcProxPlus POE for PLCs'.

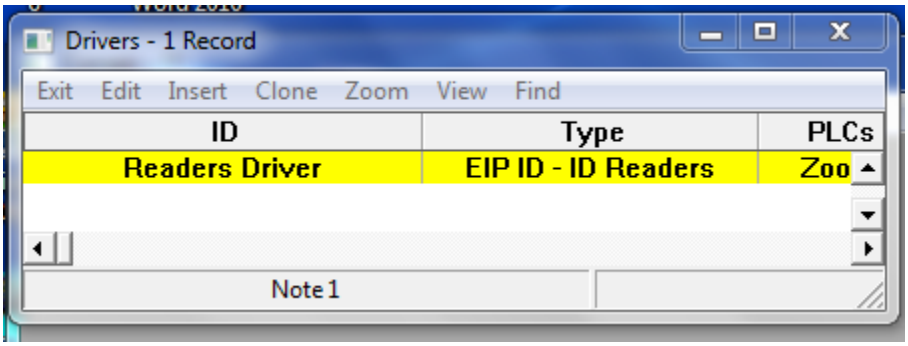
Initial Planning

The user needs to verify with RFIDeas that the card format that they are using is compatible with the reader. If the reader's configuration needs to be changed from its default values the user will have to know which settings need to be changed and the proper values for the application. The user needs to determine what IP addresses are to be used for the reader(s) and make sure that they are on a subnet compatible with the Corsair computer. This usually means that the first 3 numbers out of the 4 number IP addresses need to be the same on all the readers and on the Corsair computer. The fourth number needs to be different for each device on the network. A fourth number of 0, 1, 254, or 255 is not recommended.

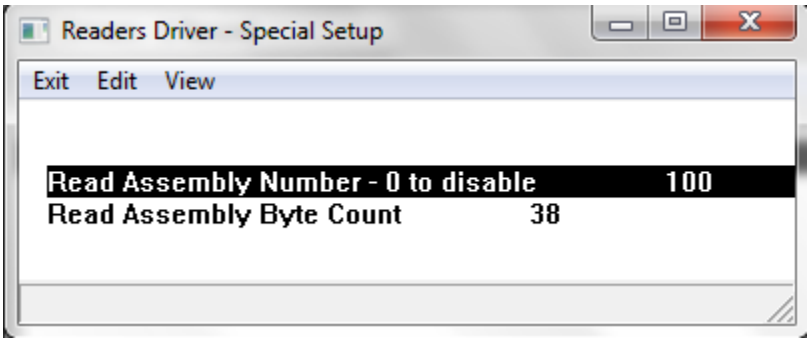
Corsair Development

The Corsair program has many features that can be used to commission and troubleshoot the readers. It is recommended that Corsair database configuration be done first. The developer level of the program should be set to 'Admin' before the work can continue.

The first step is to create a Corsair driver record. It can be given any name for an ID. The Type field needs to be set to 'EIP ID – ID Readers'.



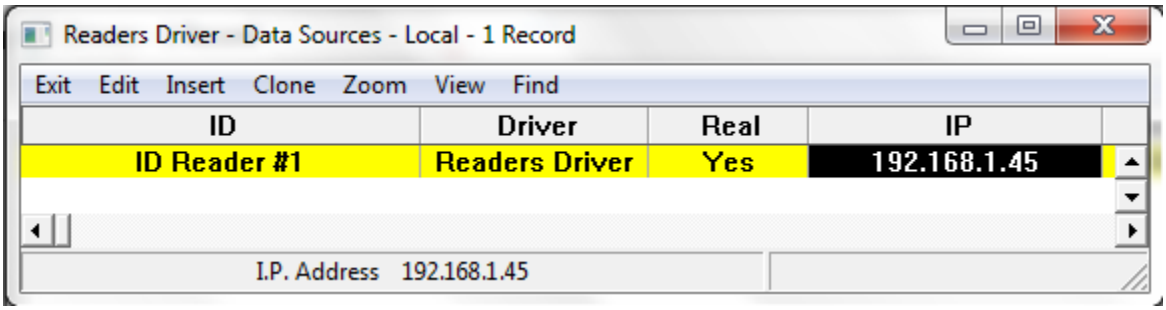
This driver has special setup parameters that must be set. Driver special data setup is accessed by zooming from any field of the driver record that is not a normal zoom field.



As of the time of this document there are only two driver parameters that must be set. 100 must be used for the Read Assembly Number and 38 as the Read Assembly Byte Count.

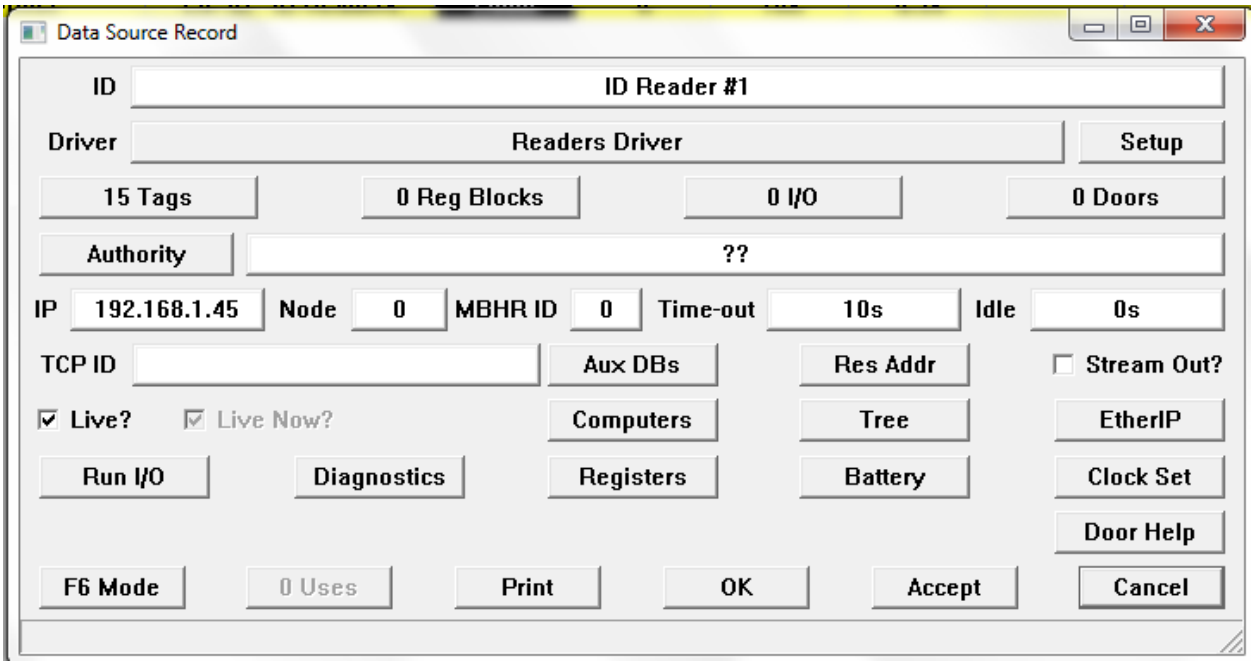
These are driver-level and not data source-level parameters. That means that they apply to each of the data sources that are on the driver. Future revisions of the program may have more options. If different readers require different options they will have to be configured under different driver records.

The next step is to create data source records under the driver. Each data source corresponds to an ID reader. Go to the Data Sources Zoom field on the driver record and zoom to the local view of sources on that driver.



Each data source must have its unique 4-part IP address entered into that column. The 'Real' field must be set to 'Yes'.

The next step is to create several tags under each data source using special addresses that have been reserved for use with this driver. The easiest way to do this is let the Corsair program create them automatically. From each data source record press 'F6' to go to single record editing. Clicking on the 'Res Addr' button starts automatic creation of the tags. As of the time of this document 15 tags will be created. Pressing F6 again returns the computer to the data sources editing window. The tag creation process will need to be repeated for each reader that has been configured into the system.



Reader Configuration

Most systems will use a subnet mask of 255.255.25.0 and a gateway of 0.0.0.0. The ACP Client Mode is to be set to 'Listen' and not to 'Poll'. These appear to be default values for these readers.

It is recommended to leave the Serial Tunnel TCP Port at the default value of 10001.

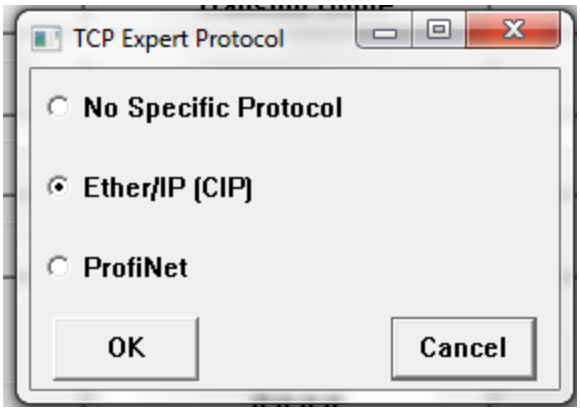
Normally the only required configuration change to the reader is setting in the proper IP address that matches the IP address that has been entered into the Corsair data source record. Corsair does not require that LUID numbers be entered into each reader. They can be left at the default value of zero.

Setting the static IP address into the reader

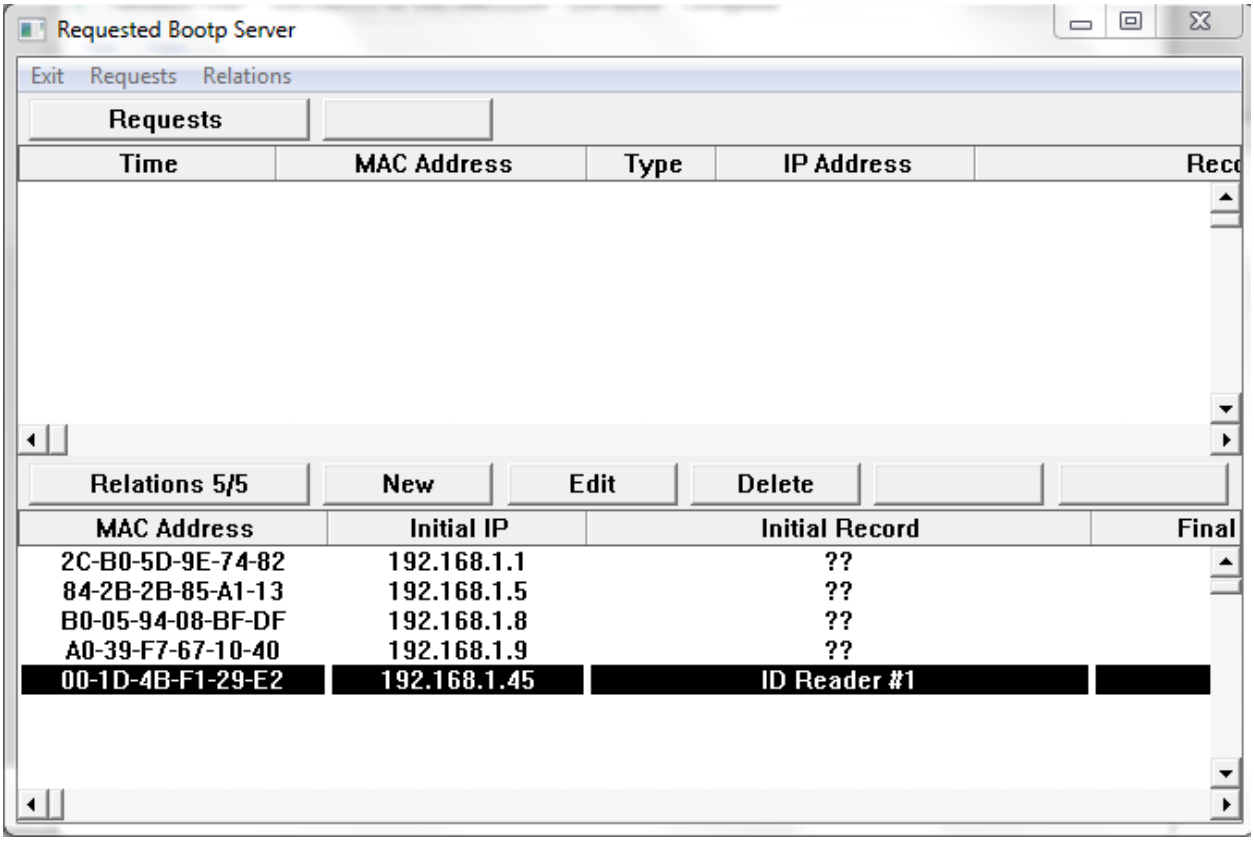
If the user has more than one reader it is recommended starting with only one of them powered and hooked into the network. Set the proper static IP address into it and then cycle its power. After verifying that it powered up with the proper address it can be left on while the second unit is powered up.

There are several ways that the IP address can be entered. The CorsairHMI program contains a BOOTP server utility that may be used. It has the capability to shut off BOOTP on the reader after the IP is set. This makes the address a static address that will remain after power is cycled. With static IP addresses it is not necessary to have BOOTP software running on the network at all times.

Corsair BOOTP is accessed through the TCP/IP expert. Clicking on the 'Protocol' button allows the selection of Ether/IP protocol. This step must be taken before the BOOTP process is started.

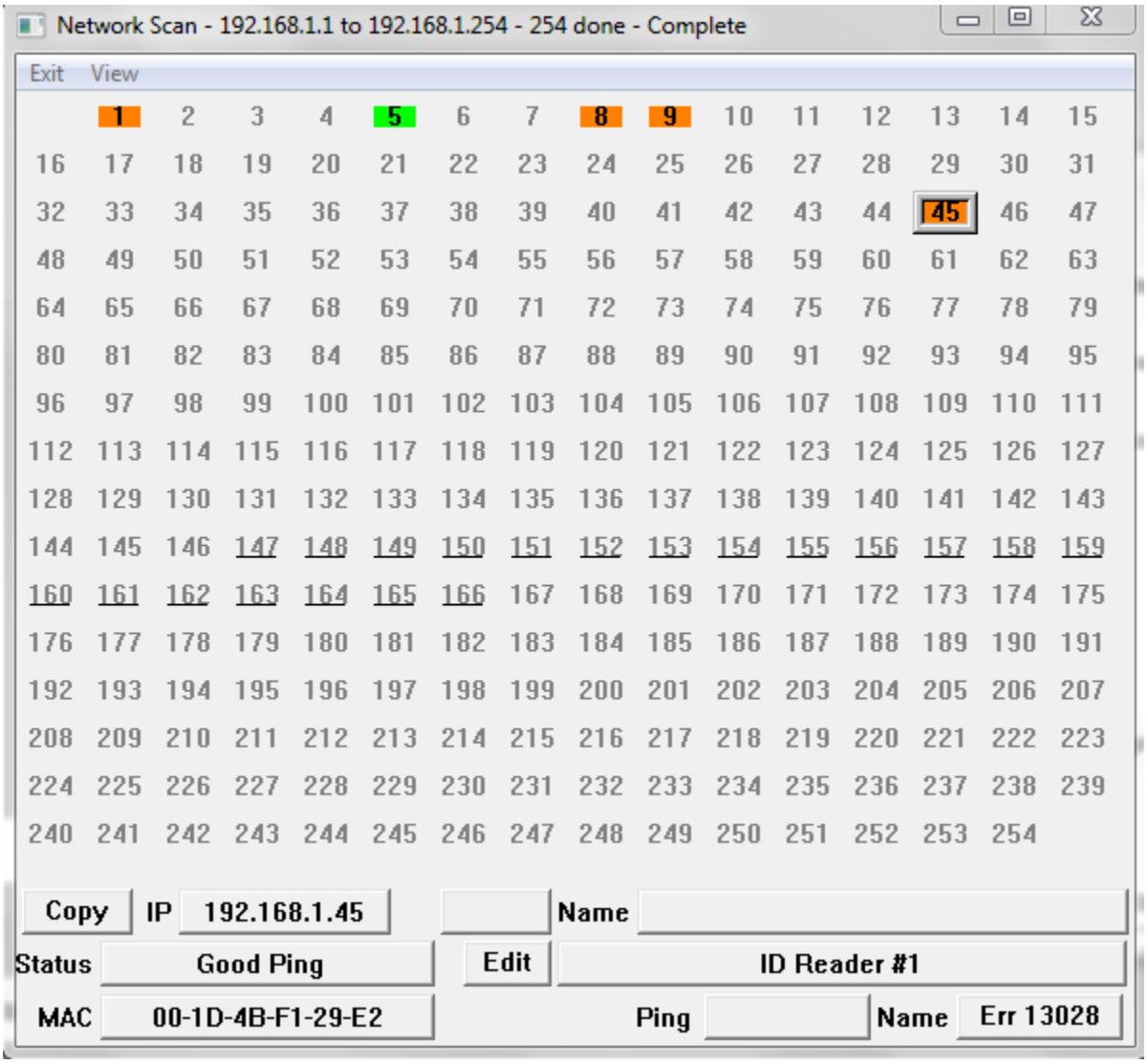


The next step is to click on the Requested BOOTP button. Other documents on the CorsairHMI website document how this window is used so that information will not be repeated here. The Ether/IP protocol selection means that BOOTP can be shut off from Corsair when the address has been set.



Corsair Ping Scan

The Corsair ping scan utility can be used to see which readers are on the network during reader configuration and afterwards for maintenance purposes. The scan window continuously sends out 'Ping' commands to each of the possible addresses on a 254-address subnet. It then shows which addresses respond to the Ping.



As a minimum, the scan should show each of the card readers in the system and the address of the Corsair computer. If the scan does not work network Firewall settings may need to be changed.

Browsing the Reader

The Corsair program can be used to start a browser to do some configuration tasks on the reader. This is possible if the user knows what IP address is being used by the reader. If there is a data source record in the Corsair database with an IP address that matches what is in the reader the following procedure will work:

Set the Corsair program for development administration.

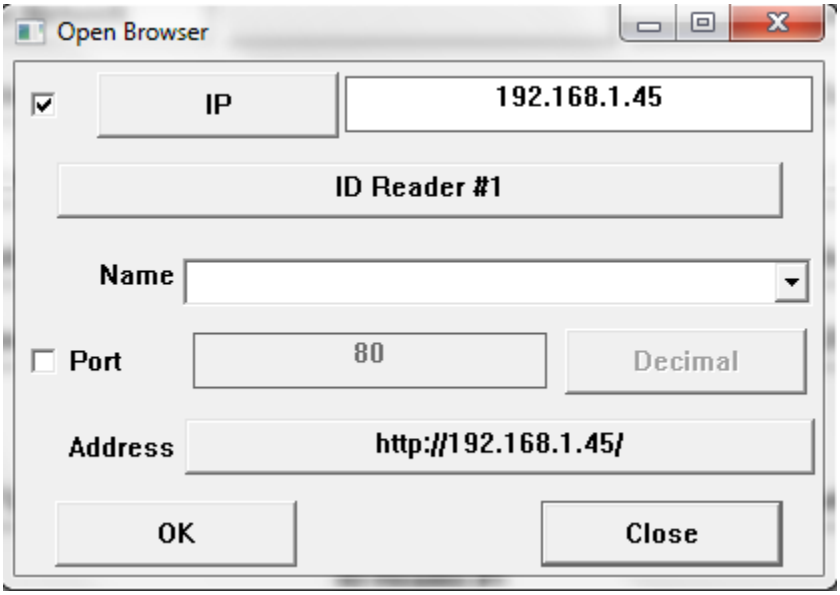
Open the TCP/IP Expert window.

Click on the ‘Browse’ option.

Turn on the ‘IP’ Checkmark at the top of the window.

Click on the ‘IP’ button to select the reader that you wish to browse.

At this point the window should look similar to this:



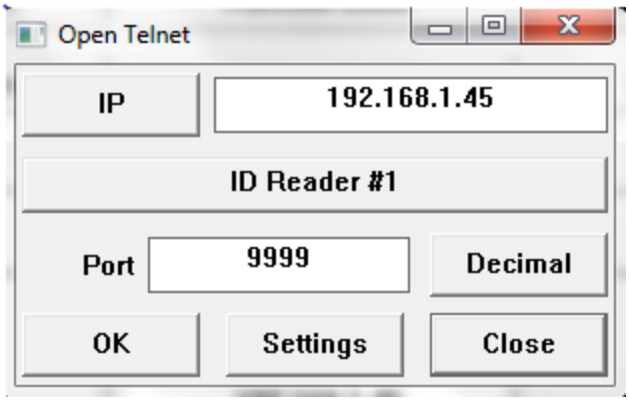
The ‘OK’ button can then be used to open a browser window to the reader. The Internal Mode setting should generally be ‘Listen for Card Reads’ and not ‘ACP QID Poll’. BOOTP should not be enabled on a reader once it has been assigned a permanent IP address. Closing this browser window, cycling the power on the reader, waiting for a minute, and then returning to this window is a possible way to check that the proper static IP address is configured in the reader.

Telnet Configuration of the Reader

The Telnet protocol can be used to configure some items and to monitor the reader. Many modern computers do not come with a Telnet client installed by default. It may already be on the computer’s hard drive and merely have to be enabled. Telnet clients default to using port 23. This port number will not work with the RFIDeas readers. They use port number 9999. A typical entry from a command line is:

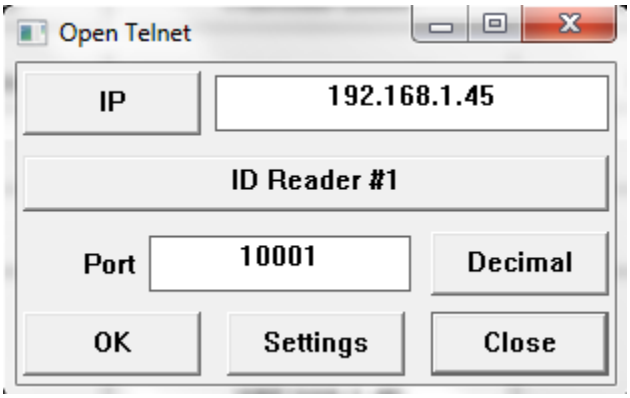
C:\>telnet 192.168.1.45 9999

An alternative is to use the Telnet utility that is part of the CorsairHMI TCP/IP expert.



The readers Telnet menu includes a Debug option that can be used to see how the reader is responding to cards. It can also be used to verify that the reader is set for Listen and not Polling mode.

The RFIDeas readers also include a serial tunnel option that can be used for more complex configuration work than what is possible with the port 9999 Telnet. It is also accessed with the same Telnet client except that it uses port 10001.



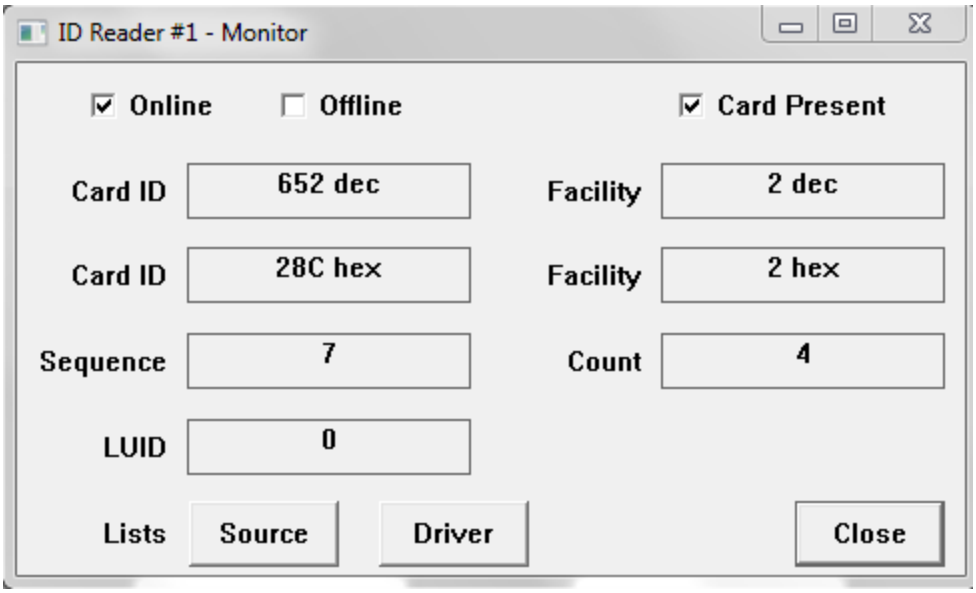
RFIDeas has an Ascii Command Protocol document that tells how to use this serial tunnel. One possible use for it is to set a LUID number into the reader.

Source Diagnostics

The next step is typically to start the Corsair interface by checking the Interface checkbox and checking the Source Diagnostics window for each reader. If everything is working the 'Good Reads' count on the window should be rapidly increasing. This count is not a count of how many cards the reader has scanned. It is a count of how many times the Corsair program has communicated with the reader.

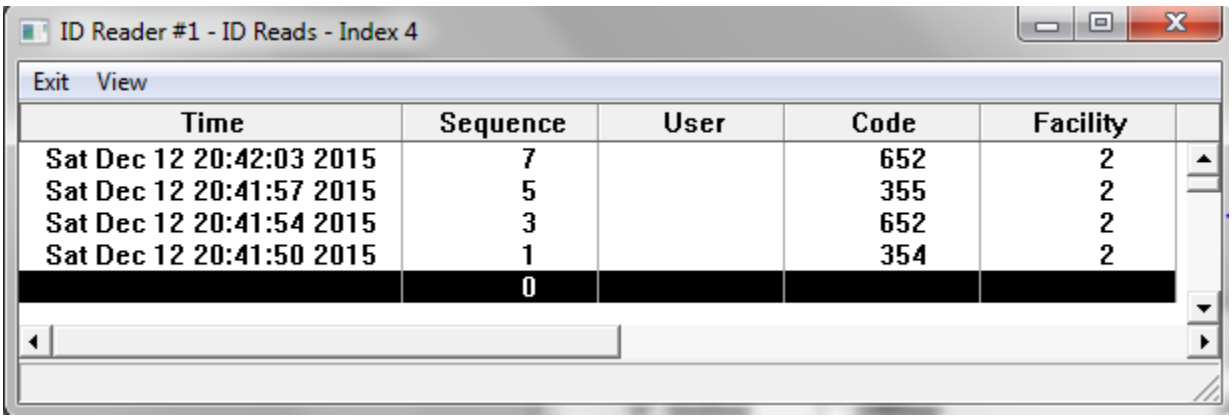
Register Monitoring

The driver includes a register monitor function that is used to see what the reader is doing.



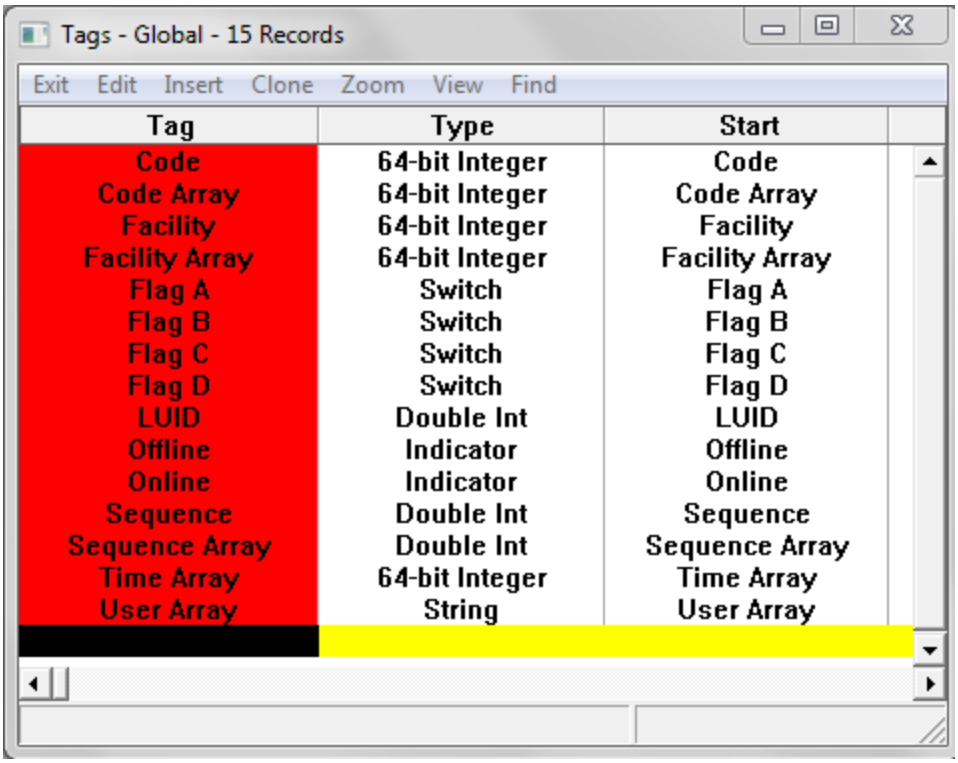
This window shows Card ID and Facility code in both decimal and hexadecimal. The count is the number of cards that have been read since the Corsair Interface started running. The sequence number is incremented by one when a card is put on the reader and incremented again when the card is pulled away.

The 'Source' List button on the register monitor is used to open a list of reads for this device.



Driver Addresses

The Corsair program generated these tags automatically.



The 'Flag A' through 'Flag D' switches are turned on by the driver when the card is first read by the reader. They can be used for several purposes. The most common purpose it to trigger a script. The script needs to turn the switch off to prepare it to detect the next card read.

Orion BMS

CorsairHMI can be used to communicate with the Orion Battery Management System from Ewert Energy Systems. The connection is made through Ewert's CANdaptor module. Ewert has software specifically made to work with the Orion. They also recommend products to use for a 'dashboard' for the Orion. Corsair can have an advantage in Orion systems that also need to work with a wide variety of devices needing more communication options than what is available with other programs.

The Orion BMS can transmit data through the CANbus using regularly transmitted data messages or by using OBD2 active data polling. The Corsair-Orion interface uses regularly transmitted data messages. This is one-way transmission. The Corsair program does not change any data on the Orion. It passively receives data.

The driver record for the Orion system must have the type of 'Ewert Can – CANbus'. It needs to be assigned a port number corresponding to the serial port used by the USB CANdaptor module. The CorsairHMI Designer manual has general information about the configuration of this driver that will not be repeated here.

The developer will probably want to configure the Orion to send the Battery Cell Broadcast message. This must be done with respect to Ewert's guidelines about CANbus traffic. Corsair can use this message

to determine Cell Instant Voltage, Internal Resistance, Open Voltage, and shunting status. Corsair will not use the checksum that is at the end of the message.

The ID for the cell broadcast is set in the Orion configuration. For this example assume that it is hexadecimal 36. A data source must be created under the driver record. It must be set to be 'Live'. This data source will only be used to receive cell broadcasts. It's frame identifier needs to be set to '11-bit standard'. 'Use ID Filter' needs to be set to 'Yes'. The ID Filter Mask needs to be set to 7FF. The Match Minimum and Maximum values both need to be set to 36.

The first byte of the cell broadcast message is the Cell ID number. The ID for the first cell is zero, for the next cell is one, and so on. Tag addresses for each cell must begin with a 'BYTM' byte match instruction. For the first cell it is 'BYTM 0, 0', for the next cell it is 'BYTM 0,1', and so on.

The next two bytes of the cell broadcast message are the Instantaneous voltage of the cell. This requires an integer tag. The full address for the voltage of the first cell is

BYTM 0,0; BYTP 1,2

The full address for the instant voltage of the second cell is

BYTM 0,1; BYTP 1,2

The open voltage for the first cell is an integer tag at

BYTM 0,0; BYTP 5,2

The remaining cell items are the internal resistance and the shunting status. The internal resistance requires a tag with the 15-bit field data type. Its address is

??

The shunting status requires a tag with the Switch data type. Its address is

??

More data sources must be set up under the driver to receive messages that are not cell broadcasts. The Filter ID Mask must be set on each one for the CANbus ID that it is supposed to receive. Typically, the BYTM byte match instruction is not needed for tags on these data sources.

Raspberry PI Systems

A Corsair program that is designed to run on the PI will be a file named 'corsair32pi'.

Corsair currently requires 4 packages that must be installed before it can run. Suggested Linux terminal commands are:

'sudo apt-get install libmicrohttpd-dev'

'sudo apt-get install unixodbc-dev'

'sudo apt-get install libcurl4-gnutls-dev'

'sudo apt-get install libgtk-3-dev'

Do not type the single quotes.

If the program is downloaded from the Internet or copied to the PI from a USB stick it may need for its permissions to be modified before it can run. This is typically done from the Linux terminal mode. Change to the same directory as the one that the program is in. Type:

'sudo chmod a+x corsair32pi'

Remember that Linux is case-sensitive.

Corsair model files typically have a '.cap' extension. They frequently may be copied from a Windows machine to a PI and back with no required changes. The exception is anywhere there is a file path specification since the syntax for them is different between Linux and Windows. The computer properties (.cfg) file can rarely be copied between Linux and Windows because it contains the path specification for model files and names for serial ports.

I2C Interface Examples

The Linux I2C driver is used to interface integrated circuits to the PI processor. Corsair currently only supports input data over I2C where the chip gives data to Corsair. It cannot be used for cases like analog outputs where Corsair sends variable data to the chip.

This example assumes that two devices are hooked on the I2C buss. The first is a TMP102 temperature sensor. The second is an ADS1015 analog to digital converter. If the ADDR pin of both devices is left unhooked they will probably both be at address 0x48 so there will be a conflict. Setting the ADDR pin of the ADS015 to the positive supply will move it to address 0x49 and eliminate the conflict.

Once the hardware is set up the 'i2cdetect' command is used to see if the chips are communicating at the correct addresses. Responses should be seen at both 0x48 and 0x49.

Two driver records are created. Both are the 'Linux I2C' type. The special data on both of them starts with '/dev/i2c-1' as the file path and name.

The TMP102 temperature sensor will be read with the simplest possible mode. Bus write and read addresses on the driver are set to 0x48. There are no commands under the driver.

There is one data source under the temperature driver. It is named 'Temperature Reading'. It has an idle pause time of 0.25 seconds.

The setup data under the data source specifies 0x48 for the bus write and read addresses. Command 1 is 'Read Bytes' with a value of 2. It is the only command so Command 2 is '??'.

The data source contains only one tag named 'Temperature'. Its type is 'Float' with a -3.1 format. The start address is:

RBYTP 0,2;SHR 4; INT 12; MUL 9;DIV 80;ADD 32

The RBYTP reverses the two bytes of data to match the byte-ordering convention that the PI uses where the least-significant byte is first. The SHR instruction shifts out unused bits. The INT instruction tells Corsair to do sign-extension on a 12-bit integer.

The temperature from the chip is a 12-bit value expressing temperature in $1/16^{th}$ of a degree Centigrade. It is desired to show it in Fahrenheit. The integer value must be divided by 16 and then taken times 9 divided by 5 plus 32 to get to Fahrenheit. The divide by 80 combines the divide by 16 and the divide by 5.

The A to D converter takes the second driver. All bus read and write addresses are set to 0x49. There are two data sources. Each is used to read a different channel of the converter. The first data source is named 'First Channel'. The second data source is named 'Second Channel'. The Idle Pause times of the sources are set to values that are adequate for the application but not so fast as to take excessive CPU time.

The instruction sequence for the first channel is:

Command 1 Write Byte Value 0x1 Point to config register

Command 2 Write Byte Value 0x4 Config register bits 8-15

Command 3 Write Byte Value 0x83 Config register bits 0-7

Command 4 Finish Write

Command 5 Write Byte Value 0x0 Point to conversion register

Command 6 Delay Value 2 Time for conversion

Command 7 Read Bytes Value 2 Get conversion register bytes

Command 8 ??

The instruction sequence for the second channel is:

Command 1 Write Byte Value 0x1 Point to config register

Command 2 Write Byte Value 0x34 Config register bits 8-15

Command 3 Write Byte Value 0x83 Config register bits 0-7

Command 4 Finish Write

Command 5 Write Byte Value 0x0 Point to conversion register

Command 6 Delay Value 2 Time for conversion

Command 7 Read Bytes Value 2 Get conversion register bytes

Command 8 ??

These instruction sequences are identical except for the setting of the MUX input multiplexer bits. They determine what ADS1015 input pins are read. Bit combinations can be set to change the PGA gain for each input. The chip manufacturer's data sheet needs to be consulted to determine the values needed for Commands 1 and 2.

Tags on the data sources for the ADS1015 analog input should have addresses that start like this:

RBYTP 0,2;SHR 4; INT 12

This reverses the order of the two data bytes in the conversion register, shifts out 4 unused bits, and recognizes 12 bits of signed integer. The rest of the address typically consists of math instructions for scaling.

New Generation Motors

The New Generation Motors EVC402 motor controller communicates using a nonstandard serial protocol. CorsairHMI and other industrial software can communicate with it if the serial data is translated to the industry standard Modbus protocol.

This document specifies a program that can be developed to run on an inexpensive controller. It translates between Modbus and EVC402 protocol. The resulting system is referred to as an NGM EVC402 Modbus Translator – abbreviated as NEMT.

The NEMT must have two serial ports. One connects to the Corsair computer which uses a Modbus RTU protocol driver. Corsair acts as a master and the NEMT acts as a slave. The NEMT port connects to the EVC402 motor controller. The NEMT acts as a master and the EVC402 acts as a slave. All communications are at 19200 baud with 8 data bits, 1 start bit, and no parity.he character format is 8 data bits, 1 start bit, and no parity. There is no flow control. The NEMT talks to the EVC402 with 2 stop bits. Corsair talks to the NEMT with 1 stop bit.

Both protocols use a simple Query-Response format. The normal sequence is:

Corsair sends a Query to the NEMT.

The NEMT sends a Query to the EVC402.

The EVC402 sends a Response to the NEMT.

The NEMT sends a Response to Corsair.

EVC402 Register Pages

EVC402 registers are identified with 3-digit hexadecimal numbers. The first digit ranges from 0x0 to 0xC for a page number. The remaining two digits identify the offset within the page. The resulting register address can be translated to a decimal address.

For example, the first Instrumentation register is 0x100. This is the Motor speed in RPM. The decimal equivalent of this address is 256. The second Instrumentation register is 0x101. This is the Supply Voltage at decimal 257.

Each EVC402 register has 16 bits. Within the register these are called bits 0 through 15 with 0 as the least significant bit and 15 as the most significant. The address of a bit is the register address multiplied by 16 added to the 0-15 bit offset.

The EVC402 Drive State register is at hex 0x10C. This corresponds to a decimal register address of 268. Bit 0 shows that the drive is operating in reverse. Its bit address is 268 * 16 + 0 or 4288. The speed control indication bit is bit 1 of the same register. Its bit address is 4289.

Modbus Address Translation

The 16-bit integer registers of the EVC402 correspond directly to Modbus 4X Holding Register references. One must be added to the decimal address and the result prefixed with 4. Typically zeros are padded to show a total of 6 digits. The address 256 Motor Speed in RPM is at Modbus address 400257. The Supply Voltage is at Modbus address 400258.

Individual bits of the EVC402 registers correspond to Modbus 0X Coil references. One must be added to the decimal bit address and the result prefixed with 0. Typically zeros are padded to show a total of 6 digits. The speed control indication bit is at Modbus address 004290.

Level 1 Modbus Function Codes

Level 1 defines the minimal level of features that are required for an NEMT to be used with a Corsair system. Modbus Queries include:

Function Code 1 – Read Coil Status – with a point count of 1

Function Code 3 – Read Holding Registers – with a point count of 1

Function Code 15 – Force Multiple Coils – with a point count of 1

Function Code 16 – Preset Multiple Regs – with a point count of 1

Function code 3 translates to a Query command to the EVC402. Function code 16 translates to an Assignment command to the EVC402. Function code 1 translates to a Bit Query. Function code 15 translates to a Bit Assignment.

Transmitted Character Conventions

The Modbus side of the NEMT will not use any of the timing conventions that are defined for the Modbus RTU protocol. Corsair will send a Query and then wait an adjustable time for a response.

The NEMT can query the EVC402 expecting either decimal or hexadecimal replies. Hexadecimal replies are preferred. Characters transmitted to the EVC402 shall all be upper case. This includes the A through F hexadecimal digits. The NEMT shall accept responses from the EVC402 in either upper or lower case.

The NEMT will terminate all transmissions with a single Carriage Return (decimal 13) character. The EVC402 is expected to echo back [CR][LF]. LF is the Line Feed (decimal 10) character.

Communications Sequencing

It is essential that each communication is completed before another is attempted. The Corsair timeout must be adjusted high enough that the NEMT and EVC402 have enough time to complete any exchange before Corsair starts another. The timeout value for NEMT to EVC402 communications is up to the NEMT programmer. It will probably have to be determined experimentally.

When the NEMT is expecting a reply from the EVC402 it will not act upon that reply until it sees the terminating CR carriage return. It may then transmit the response to Corsair. It does not have to wait for a LF line feed character.

The NEMT will discard all LF characters coming from the EVC402.

Modbus Slave Address

Every Modbus RTU communication starts with a one-byte slave address that can range from 0 to 255. The NEMT is to respond to any slave address. The address value is not used for determining required communications to the EVC402. It is essential that every response from the NEMT to Corsair uses the same slave address as Corsair sent in the query. Corsair reserves the right to use different slave addresses for different Queries. The NEMT must always use the correct response.

Level 2 Enhanced Capabilities

Level 2 capabilities make the NEMT more usable in general-purpose industrial work. They do not offer any additional advantages when used with CorsairHMI.

Level 2 would add the following Modbus function codes:

Function Code 2 – Read Input Status – with a point count of 1

Function Code 4 – Read Input Registers – with a point count of 1

Function Code 5 – Force Single Coil

Function Code 6 – Preset Single Register

Function code 2 would be treated by the NEMT exactly like Function code 1 except for the different code number. This means that Modbus 1X references map to EVC402 bits the same way as Modbus 0X references do. Function code 4 would be treated by the NEMT exactly like function code 3 except for the different code number. This means that Modbus 3X references map to EVC402 registers the same way as Modbus 4X references do. In all non-exception cases the NEMT response function code must match the Corsair Query function code.

Level 3 Enhanced Capabilites

A desirable feature is for the NEMT to be able to handle point count values greater than 1 for Modbus function codes 1, 2, 3 and 4. The NEMT could then use the P Page Query command. Corsair allowed read register lengths must be adjusted so that Corsair never tries to read registers on two different EVC402 pages within a single query.

A level 3 NEMT would enable considerably faster system performance.

Another Level 3 capability would be adding Modbus Function Code 22 – Mask Write 4X register. There is some performance to be gained with this function if Corsair is properly configured.

Level 4 Enhanced Capabilities

A level 4 NEMT would utilize buffering. It would read entire pages of data from the EVC402 using the Page Query commands with a very short time between reads. When it receives Queries from the Modbus it would then reply with values that it has in its memory instead of issuing a separate Query to the EVC402. The Page Query polling cycle will have to be suspended temporarily whenever a register or bit write comes from the Modbus. It would then be resumed as soon as the write was completed.

EVC402 Commands

Special code must be written in the NEMT for handling EVC402 Serial Control Commands. This is using holding register address 5000 which corresponds to Modbus address 405001. If Corsair tries to read this register using Function 3 the NEMT immediately returns a value of 0 without any communications to the EVC402. If Corsair writes it using Function 16 the value that it is trying to write is the value for a controller command. If it writes a 2 to Modbus 405001 the NEMT sends a 002![CR] command to the EVC402. It then waits for the controller to return an error code. After that it sends a reply to Corsair with exception responses for errors if needed.

Exception Responses

Errors in communications with the EVC402 will result in Modbus exception codes being returned to Corsair. A Modbus exception response is sent with the most-significant bit of the function code turned on. Errors should be translated as follows:

EVC402 Error	Modbus Exception
Timeout (no response)	06 Slave Device Busy
0x01	01 Illegal Function
0x02	02 Illegal Data Address
0x03	01 Illegal Function
0x04	03 Illegal Data Value
0x05	07 Negative Acknowledge
0x06	07 Negative Acknowledge
0x07 – 0x0B	01 Illegal Function
0x0C	02 Illegal Data Address
0x0D – 0xFF	01 Illegal Function

Corsair Tagging

The normal Modbus RTU serial driver can be used with the NEMT. The Modbus serial timing driver may be a better choice as it offers the developer better control of polling intervals on different data items. Run-time instrumentation typically needs to be scanned at a faster rate than configuration information.

Most integer tags developed for the EVC402 will have '-5.0' data format. If the register is shown in the NGM documentation as deci-units (like deci-amps) '-4.1' may be the correct choice for the format.

Streaming Serial Packet Protocol

CorsairHMI has multiple ways of implementing Streaming Serial data security. One version involves one computer streaming Corsair tag data to another computer. Both computers have similar Model databases. The receive computer uses the serial streaming driver. This driver differs from other drivers in that it uses the MBHR address for each tag instead of the normal starting address. MBHR addresses must be compatible with the Modbus protocol.

The data packet that is transmitted to this driver is unique to the CorsairHMI program. It is not a standard Modbus message. It does have similar characteristics to a Modbus ASCII mode message. Because of this programmers that are familiar with Modbus ASCII will have little trouble understanding the format of the data packet.

The driver can use characters with either 7 or 8 data bits, 1 or 2 stop bits, and any parity. Both computers must be set for the same character format and baud rate. 7 data bits with 1 stop bit and even parity is a suggested preference.

Like with Modbus ASCII, the protocol sends a byte of data as two consecutive hexadecimal characters. A data byte may contain a decimal value of 254. This corresponds to hexadecimal FE. It would be transmitted as the character 'F' followed by the character 'E'.

The packet is formatted with 8 fields as follows:

Field 1 – Header – 1 character

Field 2 – Slave Address – 4 bytes, 8 characters

Field 3 – Function –1 byte, 2 characters

Field 4 – Starting Address – 4 bytes, 8 characters

Field 5 – Data Byte Count – 1 byte, 2 characters

Field 6 – Data – 1 to 250 bytes, 2 to 500 characters

Field 7 – Error Check – 1 bytes, 2 characters

Field 8 – Trailer – 2 characters

The Header field is a single character. It is fixed at an ASCII colon (:) having a hexadecimal value of 3A.

The Slave Address field is a 2-byte field instead of the single byte used in normal Modbus communications. It can vary from 1 to 65535. Zero is never used for the slave address. The most significant byte is transmitted first followed by the least significant byte.

The Function field can contain 4 possible values to denote the type of the reference address. The following values are possible:

Function	Data Type	Reference
1	Discrete Output, Coil	0X
2	Discrete Input	1X
3	Holding Register	4X
4	Input Register	3X

Any other value in the function field invalidates the entire packet.

The Starting Address field is 4 bytes instead of the two that are used with normal Modbus. This permits a much larger range of address values. It is transmitted most-significant byte first and least-significant byte last.

The Data Byte Count field is a single byte with values ranging from 1 to 250. It reports the byte (not character) length of the Data field.

The Data field contains dynamic data. It is arranged as per normal Modbus. Registers are sent most-significant byte first followed by least-significant byte.

The Error Check field is a standard Modbus ASCII LRC check. It is calculated over the bytes in fields 2 through 6 inclusive.

The trailer field is an ASCII carriage return (decimal 13) character followed by an ASCII line feed (decimal 10) character.

Writing Modbus Communications

A programmer may wish to write an implementation of the Modbus protocol to communicate between two devices. Corsair can act as a master using Modbus ASCII serial protocol. It can act as a master or a slave using Modbus RTU serial protocol. It can act as a client or a server using Modbus TCP protocol over Ethernet. These recommendations for Modbus implementation can serve the programmer that wishes to communicate with Corsair or with other devices.

When implementing a slave or server the programmer will typically have a register map of the data that is available. Packing this data together into adjacent registers helps to make communication more efficient. 125-register reads and 100-register writes should be allowed. Bit reads should allow a length of 2000. Bit writes should allow a length of 800. If the master or client sends a read request of unimplemented registers or bits the slave should return zeros instead of an exception.

The 3.5 character time delay that is part of the RTU protocol can be very difficult to implement on most hardare. Many implementations do not use it.

Standard Modbus allows the addressing of over 65000 holding registers. If a system requires more data to be passed than this the programmer may be tempted to use 6XXXXX Extended Memory references. It may be more desirable to use multiple 'banks' of 4XXXXX Holding Registers with different Slave Addresses. The Slave address is a one-byte address that can vary from 0 to 255. It is best to avoid address 0 and addresses over 247. The 247 addresses from 1 to 247 should be adequate for most purposes.

Slave addresses 0 and 248-255 have been defined in the past for broadcast communicatons. Serial broadcast is unusual and should be avoided if possible. Corsair has no provision for sending or responding to serial broadcasts.

If a programmer implements server-side Modbus TCP communications are routed using IP addresses. The Query packets contain the one-byte slave address. If there is no need for more than one slave address the server should respond to any slave value and replies should echo whatever value it is sent.

Serial slaves should respond to only one slave address so that they can be used in multi-drop situations.

TCP server implementations should allow multiple simultaneous connections. They should default to TCP port 502 but be adjustable to use other port numbers.

TCP server implementations should allow pipelining where the client sends additional Queries before the server sends a reply. This can make a large difference in data throughput for some systems. It can create serious problems in some systems with inadequate CPUs and little buffer memory. If a programmer writes a client that can pipeline the pipeline count should be adjustable from 1 to no more than 16.

32-bit integer and float data should follow the byte-ordering conventions of a Modicon PLC. Adjustable byte-ordering is desirable. Corsair uses fixed X86-style byte ordering for 64-bit integers and floats.

If a programmer implements a slave side he needs to consider data consistency issues. If a master requests a single 4x register that maps to either half of a 32-bit floating point value he may elect to return an exception like 02 – Illegal Data Address.

ASCII serial implementations should always terminate messages with 'Carriage Return – Line Feed'. Messages can be accepted with just Carriage Return without the Line Feed. The A through F in hexadecimal characters should be transmitted in upper case (capitals) but received in either case.

The following function codes are not considered to be Modicon-specific and so they may be used:

01 Read Coil Status

02 Read Input Status

03 Read Holding Registers

04 Read Input Registers

05 Force Single Coil

06 Preset Single Register

15 Force Multiple Coils

16 Preset Multiple Regs

22 Mask Write 4X Register

23 Read/Write 4X Registers

Modicon-specific function codes should be avoided. These include:

07 Read Exception Status

08 Diagnostics

11 Fetch Comm Event Counter

12 Fetch Comm Event Log

17 Report Slave ID

These funcitons should be avoided:

20 Read General Reference

21 Write General Reference

24 Read FIFO Queue

Ideally slave systems that receive a function code that is not implemented should return Exception Code 01 Illegal Function.

In some slave systems it may be desirable for the master to access the same data as 3XXXX Input Registers with function code 4 or as 4XXXX Holding Registers with function code 3. A read of 30007 or 40007 would get the same result. This accomodates limited systems that cannot read Input Registers. Writes must go to Holding Registers.

Implementing function code 16 Preset Multiple Registers with a count of 1 may be preferable to implementing function code 06 Preset Single Register. Corsair will respond to either.

Implementing function code 15 Force Multiple Coils with a count of 1 may be preferable to implementing function code 05 Force Single Coil. Corsair will respond to either.

Many devices and Corsair do not respond to function code 23 Read/Write 4X Registers. Separate reads and writes are usually best unless the target is an I/O device specifically designed to use the function.

Many devices do not respond to function code 22 Mask Write 4X Register. The alternative is a read-modify-write sequence. Whenever possible the function 22 write should be used as it is much safer and more efficient than read-modify-write.

Systems that communicate with Corsair over Ethernet that require large amounts of data may use Corsair's extended protocol. Arrangements can be made to get a specification.

AB Addressing Conventions

This is a suggested convention for developing Corsair tags with a processor that uses PLC-5 addresses. This includes the SLC and Micrologix cpus. It can include Compact and Control Logix when using the PCCC protocol. They all use data files described by a file number, a data type, and an array of elements of that type.

Corsair shall not read or write any data with a file number less than 20. Each file shall be created with a size at least 10 elements longer than what is required for the project.

The suggested file list is:

N20	Peer Communications Integer and Bit Data
F21	Floating Point Setpoints
F22	Setpoint Minimums
F23	Setpoint Maximums
F24	Other Floating Point Values
N25	HMI Read-Only Integer and Bit Data
N26	HMI Changeable Integer and Bit Data
N27	Corsair 4-bit Alarms

The N20 integer block is used for peer communications between PLCs. Data transfers should be no less than 10 elements long. They should be bidirectional. The first register of each Read or Write block should be used for communications failure detection. This uses the sequence number incrementing scheme that is described in the CorsairHMI 'PLC Class Outline'. It allows for detection of missed communications or a halted processor in either direction. Extra unused registers need to be allocated at the end of each block in case expansion is needed. Corsair will not read or write N20 data.

All setpoint values that are entered from the HMI will go to the F21 floating point data. Each element of this array will be bound to be between the values in the F22 minimum array and the F23 maximum array. Corsair will have a data sheet showing the name of each setpoint, the entered value, the minimum, and the maximum. Initially Corsair can be used to enter the minimum and maximum values but during startup they should be fixed to constants in the PLC code. These constants should gurantee that the setpoint stays within a safe range as defined by the equipment manufacturer.

The PLC code that loads the F22 and F23 constants should be executed at frequent intervals with 10 seconds being a reasonable value. The code that bounds the setpoints should be executed every scan doing one value on each pass. Assume that there are 10 setpoints and N7:0 is used as the pointer. The ladder or ST code should do this:

```
N7:0 := N7:0 + 1 ;  (* Increment the pointer *0

IF (N7:0 >= 10) OR (N7:0 < 0) THEN  (* Bound the pointer *)

        N7:0 := 0 ;

END_IF ;
```

```
IF F21:[N7:0] < F22:[N7:0] THEN  (* Check Minimum *)
        F21:[N7:0] := F22:[N7:0] ;
END_IF ;
IF F21:[N7:0] > F23:[N7:0] THEN  (* Check Maximum *)
        F21:[N7:0] := F23:[N7:0] ;
END_IF ;
```

If an invalid value like a negative gets into the N7:0 pointer the bounding rung will fix it before the indexed addressing instructions. This prevents a processor fault from an invalid index.

The F24 array is for other floating point values that Corsair needs to see. Calculation results that are not needed by the interface should be in other files like F8. The PLC programmer may elect to put file-level protection against writes of the data in this file.

The N25 array is for integer and bit values that Corsair needs to see but does not need to change. PLC internal data that is not needed by the interface should be in other files like B3 and N7. The PLC programmer may elect to put file-level protection against writes of the data in this file. The PLC battery low indication goes here.

The N26 array is for integer and bit values that Corsair needs to both read and write. This includes tags for buttons, switches, and HOAs. HOA tags should use the three least-significant bits of an N26: register. The other 13 bits of that register should not be used. There generally will not be any changeable integers in this block because changeable values are best in F21. An exception is the 15-integer time-of-day clock interface tag that Corsair requires. It must be located here. It is described in the 'Clock Logic' section of the CorsairHMI manual.

The N27 block is used for an array of Corsair 4-bit alarms. They are indexed like this:

Index 0 N27:0/0 to N27:0/3

Index 1 N27:0/4 to N27:0/7

Index 2 N27:0/8 to N27:0/11

Index 3 N27:0/12 to N27:0/15

Index 4 N27:1/0 to N27:1/3

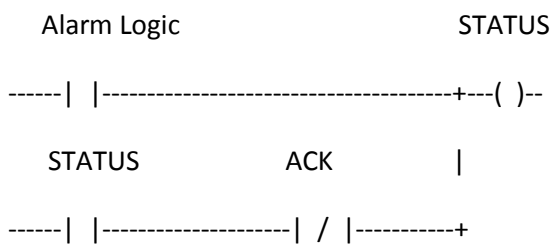
Index 5 N27:1/4 to N27:1/7

Alarms will be assigned starting at index 1. Index 0 will not be used. PLC code will be according to standard Corsair practice.

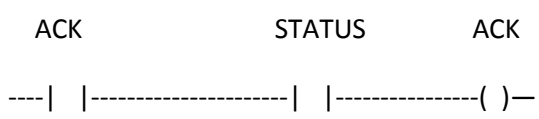
The 4 bits for alarm index 1 are:

STATUS	N7:0/4
LATCH	N7:0/5
ACKNOWLEDGE (ACK)	N7:0/6
SPECIAL	N7:0/7

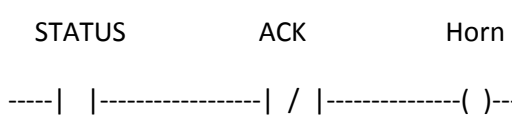
The STATUS, LATCH, and ACKNOWLEDGE bits are predefined by Corsair. The SPECIAL bit can be used for any purpose defined by the PLC programmer. The LATCH bit is often not used if the STATUS bit is latched on when the alarm occurs.



The ACK bit comes on from Corsair data communication. It shuts off when the status clears.



A horn may sound from an unacknowledged alarm.



The ‘Alarm Forms’ section of the CorsairHMI manual gives more detail as to the operation of 4-bit PLC alarms.