

CitectSCADA

Version 5.5

*Establishing Communications
and
Tagging IO Devices*

DISCLAIMER

Citect Pty. Limited makes no representations or warranties with respect to this manual and, to the maximum extent permitted by law, expressly limits its liability for breach of any warranty that may be implied to the replacement of this manual with another. Further, Citect Pty. Limited reserves the right to revise this publication at any time without incurring an obligation to notify any person of the revision.

COPYRIGHT

© Copyright 2003 Citect Pty Limited. All rights reserved.

TRADEMARKS

Citect Pty Limited has made every effort to supply trademark information about company names, products and services mentioned in this manual. Trademarks shown below were derived from various sources.

CitectSCADA, CitectHMI/SCADA, CitectFacilities and CitectSCADA Batch are registered trademarks of Citect Pty. Limited.

IBM, IBM PC and IBM PC AT are registered trademarks of International Business Machine Corporation.

MS-DOS, Windows, Windows 98, Windows 2000, Windows XP and Excel are trademarks of Microsoft Corporation.

dBase is a trademark of Borland Inc.

General Notice:

Some product names used in this manual are used for identification purposes only and may be trademarks of their respective companies.

October 2003 Edition for CitectSCADA Version 5.5

Manual Revision 1.0

Communicating with I/O Devices

CitectHMI/SCADA can communicate with any control or monitoring I/O Device that has a communication port or data highway - including PLCs (Programmable Logic Controllers), loop controllers, bar code readers, scientific analysers, remote terminal units (RTUs), and distributed control systems (DCS).

I/O Devices can be easily classified into two distinct categories for their communication connection method with CitectHMI/SCADA: - Local or Remote.

- Local I/O Devices are directly connected to a CitectHMI/SCADA I/O Server.
- Remote I/O Devices are connected to CitectHMI/SCADA via an intermediate communications means (radio link, modem and phone line, etc).

Both of these types can be configured to be permanent, periodic, or on request.

Communication Types

CitectHMI/SCADA supports four types of I/O Device communication:

- Serial communication
- PLC interface board
- Data acquisition board
- DDE Server

Whether the I/O Device is local or remote, the most common method of communicating to I/O Devices is via simple serial connection. Typically serial communications follow (to some degree) one of the three common serial standards - RS-232, RS-422 and RS-485 (outlined in the Communication Standards.)

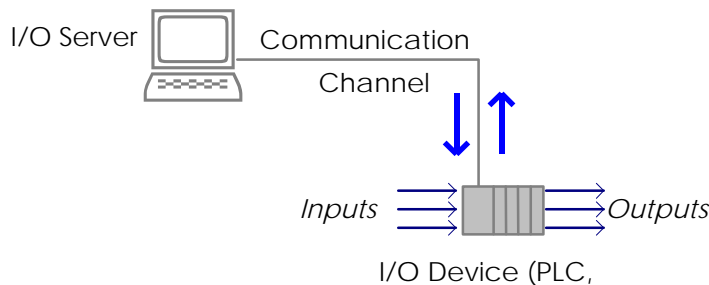
With CitectHMI/SCADA there are many I/O Device communications options - through the CitectHMI/SCADA computer's COM port, through a high speed serial board, or through a special communications board supplied by the I/O Device manufacturer. In any case, the setup of the CitectHMI/SCADA to I/O Device communications should be straight-forward - using the Express I/O Device Setup wizard.

NOTE: The majority of I/O Device communications standards are serial based. For clarity however, only simple serial communications are considered in this section. More complex serial communications, such as Ethernet, are detailed where appropriate.

How CitectHMI/SCADA Communicates with I/O Devices

CitectHMI/SCADA communicates directly with the I/O Device(s) in your plant or factory. There are three major components in this system:

- the CitectHMI/SCADA computer (I/O Server)
- the communications channel
- the I/O Device



To enable CitectHMI/SCADA to communicate with an I/O Device you need a device driver. This is the interface between CitectHMI/SCADA and the I/O Device which implements the communication protocol(s) of the I/O Device.

The structure of a CitectHMI/SCADA device driver is extremely flexible. It allows for a single driver to communicate using several hardware boards, each with several hardware ports, with each port communicating with several I/O Devices.

Inputs to the I/O Device provide information about your plant, such as the location of a product, speed of a machine, status of a drive, or temperature of an oven. Outputs from the I/O Device usually perform the tasks required to operate your plant, such as starting electric motors or varying their speed, or switching valves and indication lamps. In some I/O Devices (such as PLCs), a program stored in the I/O Device controls the outputs. The logic (control strategy) of this stored program and the status of the inputs determine the value of each output.

The value of each input and output is stored in a separate memory register in the I/O Device. Each memory register is referenced by its address.

Most I/O Devices provide a communication port or data highway for communicating with other devices or computers. By using this communication pathway, CitectHMI/SCADA can read and write to the memory registers in the I/O Device.

By reading and writing to memory registers in all your I/O Devices, CitectHMI/SCADA collects data from your plant or factory for monitoring and analysis, and provides high level (supervisory) control of your equipment and processes.

You do not usually need to read (or write to) all registers in the I/O Device, and CitectHMI/SCADA provides a project editor for you to specify the inputs and outputs that you want to monitor or control. Once you have defined these register addresses, you can use them for system control, operator displays, trend analysis, data logging and alarm indication.

NOTE: I/O Devices such as Programmable Logic Controllers (PLCs) usually have an internal program that controls the low level processes within your plant. A PLC program continually scans the input registers of the PLC, and sets the output registers to values determined by the logic of the PLC program. While you can use CitectHMI/SCADA to replace any PLC program, it is not good practice to do so. PLCs are designed for very high speed response (typically 1 to 100ms) and replacing this functionality with CitectHMI/SCADA could reduce the overall performance of your control system. You should only use CitectHMI/SCADA to complement your PLC program (i.e. for high level control and system monitoring).

How CitectHMI/SCADA Reads from the I/O Device

The computer directly connected to the I/O Device is the I/O Server for that I/O Device. The I/O Server keeps up to date information on all of its I/O Devices by retrieving data from each I/O Device at regular intervals and storing it in a cache. Then, whenever a CitectHMI/SCADA client (Display Client, Trends Server, Reports Server, etc.) requires data from an I/O Device, the I/O Server uses the information stored in the cache to provide the requested data.

NOTE: CitectHMI/SCADA uses the computer directly connected to the I/O Device as the CitectHMI/SCADA I/O Server for that I/O Device. This server may have one or more I/O Devices connected to it. For any CitectHMI/SCADA client (Display Client, Trends Server, Reports Server, etc.) to request data (status) from any I/O Device, the client requests the data from the I/O Server connected to the I/O Device, rather than directly from the I/O Device. The server retrieves the data from the I/O Device and stores it.

Before starting a task, CitectHMI/SCADA reads all the required data from the I/O Device (i.e. any data needed to complete a required Cicode task or process). For example, when you schedule a report, CitectHMI/SCADA will read all I/O Device data that the report may need, and any data a Cicode function (called by the report) may need before the first line of the report starts running.

This may have some side effects you should allow for. Firstly, as CitectHMI/SCADA must read the I/O Device when you schedule a report, or start a keyboard command etc, there will be a short delay before the Cicode starts to execute - until the read data (associated with it) is read from the I/O Device. For example, if you have a keyboard command as follows:

Key Seq:	<input type="text" value="ENTER"/>
Command:	<input ####);"="" +="" of="" plc_var"="" plc_var:="" type="text" value=""/>
Privilege:	<input type="text"/>
Comment:	<input type="text" value="Display value of PLC_VAR from the PLC"/>

CitectHMI/SCADA will first ask the I/O Server to read the value of PLC_VAR from the PLC. It will then do something else while it waits for the PLC to send the data back. When the data is returned, it will start to execute the Cicode. Because of this operation, you may even have time to start another keyboard command before this one completes. This effect could cause problems if the data was to be displayed at, say, AN 50, and while waiting for the data you changed pages. Then the value of PLC_VAR would be displayed on the new page! This is normally only a problem if you are doing some complex Cicode operations. If there is no data to be read from the I/O Device, the Cicode or process will execute immediately.

If CitectHMI/SCADA cannot read the data from the I/O Device for some reason (e.g. the I/O Device is offline or the data is bad), CitectHMI/SCADA will still start the report, Cicode etc., and generate a Hardware error. For example, **#COM** will be displayed if the variable is read from a text object returning its numeric value, but it is up to you to test for errors if it will impact on your Cicode. If you call the function ErrCom(), it will tell you if all the I/O Device variables associated with your report or Cicode are OK. So you could do the following in a report:

```
{CICODE}
IF ErrCom() <> 0 Then
    PrintLn("This Report contains bad data");
END
{END}
```

Under some special conditions CitectHMI/SCADA will not read the I/O Device before starting a piece of Cicode. This includes callback events, Alarm ON/OFF action and any Cicode task created with TaskNew() without using mode 4. Under these critical conditions, CitectHMI/SCADA wants to start the Cicode immediately, and as it is a critical operation, CitectHMI/SCADA will not read the I/O Device first. Instead, it will retrieve the data from the local copy of the variable(s). Bear in mind that there is a chance this data may be stale. If you require I/O Device data under these conditions, create a new task using TaskNew(mode 4).

How CitectHMI/SCADA Writes to the I/O Device

CitectHMI/SCADA performs writes to the I/O Device in an asynchronous manner. That is, when you write to the I/O Device, the write takes a little time to get to the I/O Device. During this time, CitectHMI/SCADA will continue to run and do other operations, like running more Cicode. If the other Cicode assumes that the write has completed immediately, then you may encounter some side effects.

If you have the following Cicode:

```
PLC_VAR = 1234;  
Prompt("Variable is " + PLC_VAR : ####);
```

The first line is a write to the PLC. When CitectHMI/SCADA executes this line of Cicode, it will generate a request to the I/O Server to write the value 1234 into the PLC variable called PLC_VAR. CitectHMI/SCADA will then execute the next line of Cicode before the PLC write has been completed. CitectHMI/SCADA does this so that the Cicode is not stopped while waiting for a slow I/O Device. As the write to the PLC has not completed, you may think that the next line of Cicode will display the last value of PLC_VAR and not the value 1234. The Cicode will display the correct value (1234) because whenever CitectHMI/SCADA writes into the PLC, it will first update its local copy of the variable - any following Cicode will get the correct value.

This solution will not work under some conditions as CitectHMI/SCADA may keep more than one copy of an I/O Device variable, and only the one associated with the current Cicode is updated. The other variables will contain the old value of the I/O Device variable until they are refreshed (with a read from the I/O Device). There is a separate data area for each display page, Cicode file, alarms, trends and reports. If you write to an I/O Device variable from a page keyboard command, the copy of the I/O Device variable associated with that page will be updated, however the copy associated with other pages and all the Cicode functions is not updated until the next read (as determined by the [Page]ScanTime parameter). If you call a Cicode function that assumes the write has completed it will get the last value.

You can work around this by writing to the I/O Device variable in the Cicode function. All Cicode functions share the same I/O Device variables so the writes will operate as expected.

```
FUNCTION  
TestFunc(INT nValue)  
    PLC_VAR = nValue;  
END
```

Another side effect of this operation is that you may think that CitectHMI/SCADA has successfully written to the I/O Device, when in fact it may later fail because of some hardware fault or configuration error, (eg write to an input register). Under some critical conditions, you may want to check that the write has in fact completed. Whenever a write fails to the I/O Device, a hardware error is generated, but the associated Cicode cannot be notified or use the IsError() function as that Cicode has executed way past the I/O Device write. You may verify a critical write by calling the function ReRead() just after the write, and then verify the value of the variable. ReRead() will force Cicode to re-read all I/O Device variables associated with the current Cicode so you can check the new value.

```
PLC_VAR = 1234;  
ReRead(1);  
IF PLC_VAR <> 1234 THEN  
    Prompt("Write failed");  
END
```

Under these conditions the data will be read from the physical PLC, not from the I/O Server cache, as the I/O Server will invalidate any cached data associated with a PLC write. This will allow you to test for a completed write. Note that other Cicode tasks running at the same time will not be waiting on the ReRead so they may see the old or new value - depending on if they are using the same copy of the PLC variable. You can also stop CitectHMI/SCADA from writing to the local copy of the variable by using the function CodeSetMode(0, 0).

Performance Considerations

The performance of any control and monitoring system is influenced by many factors. The computer, the I/O Device(s), and the communication pathway between them are obvious factors. The faster they can transfer data, the faster your system will operate. (CitectHMI/SCADA will always maintain a data transfer rate as fast as the I/O Device hardware will support.) The data transfer rate is hardware dependent - once you have selected and installed the hardware, it is effectively beyond your control.

However, there is one area where you can directly affect the performance of your runtime system - how you arrange the data registers in the I/O Device(s).

Caching Data

On large networked systems with many Display Clients, you can improve communications turn-around time by using memory caching.

When caching is enabled, all data that is read from an I/O Device is stored temporarily in the memory of the I/O Server. If another request is made (from the same or another Display Client) for the same data within the cache time, the CitectHMI/SCADA I/O Server returns the value in its memory - rather than read the I/O Device a second time. Data caching results in faster overall response when the same data is required by many Display Clients.

A cache time of 300 milliseconds is recommended. You should avoid using long cache times (in excess of 1000 milliseconds), because the data can become 'stale'.

NOTE: Do not use data caching for Memory or Disk I/O Devices.

How Data Caching Works

Data caching is designed to prevent and minimise unnecessary multiple identical reads of I/O Device data within a short period of time. Unnecessary reads can be generated when more than one client requests the I/O Server to read data from a PLC or similar I/O Device within a short (typically 300ms) period of time.

Normally, upon request from a client, an I/O Server reads status data from an I/O Device, and passes it back to the requesting client.

If the Server receives any subsequent requests from other clients before the original data is returned to the first client, it will optimize the read by automatically sending the original data back to all requesting clients. (Page General Blocked Reads shows this count).

If however, a client requests the same data immediately after the Server sent the data back to a client, the Server would then do another (unnecessary) read of the Device, that could have been serviced immediately by the previous read.

Setting the data cache time to 300ms (or similar), will prevent identical repetitive reads within that cached time frame. If further clients request the identical data from the same Server up to 300ms after the Server has sent that data to an earlier client, the cached data on the Server will be sent immediately in response to the subsequent requests.

NOTE: Multiple clients don't have to be separate CitectHMI/SCADAs on a network. They may be the alarms and trend clients in the same computer - so this optimization will affect even a single node system.

CitectHMI/SCADA also uses read ahead caching. When the data in the cache is getting old (close to the cache time), the I/O Server will re-request it from the I/O Device. This optimises read speed

for data that is about to be re-used (very frequent). To give higher priority to other read requests, the I/O Server will only request this data if the communication channel to the I/O Device is idle.

Keeping a Permanent Record of the Data

In order to keep a permanent copy of cached device data, you can save the I/O Server's cache to the hard disk. Every [IOServer]SavePeriod, the data is saved to Persistence Caches, one for each cached device.

Saving the data to disk ensures that you can shut down and restart the I/O Server without having to contact each I/O Device again to get its current values. Instead, you can simply read the values from the device's Persistence Cache.

NOTE: When **Read-through** caching is enabled for a remote or scheduled I/O Device, the Persistence Cache for that device is saved to disk when the active I/O Server disconnects from the device. This will occur regardless of the value set in [IOServer]SavePeriod. You can enable Read-through caching by setting the parameter [Dial]ReadThroughCache.

Grouping Registers

When you configure a CitectHMI/SCADA system, you must define each variable (register address) that CitectHMI/SCADA will read when your system is running. When your runtime system is operating, CitectHMI/SCADA calculates the most efficient method of reading registers. CitectHMI/SCADA optimises communication based on the type of I/O Device and the addresses of the registers.

When CitectHMI/SCADA requests data from an I/O Device, the value of the register is not returned immediately. An overhead is incurred. This overhead (associated with protocol headers, checksum, device latency, etc) depends on the brand of I/O Device, and is usually several times greater than the time required to read a single register. It is therefore inefficient to read registers individually, and CitectHMI/SCADA usually reads a contiguous block of registers. Because the overhead is only incurred once (when the initial request is made), the overhead is shared across all registers in the block, increasing the overall efficiency of the data transfer.

However, reading a block of registers where only a small percentage of the block are actually used is also inefficient. If the registers that your CitectHMI/SCADA system will read are scattered throughout the memory of your I/O Device, excessive communication will probably be required. CitectHMI/SCADA will have to either read many contiguous blocks (and discard the unused registers), or read registers individually. This additional overhead means that the system will not operate at peak performance.

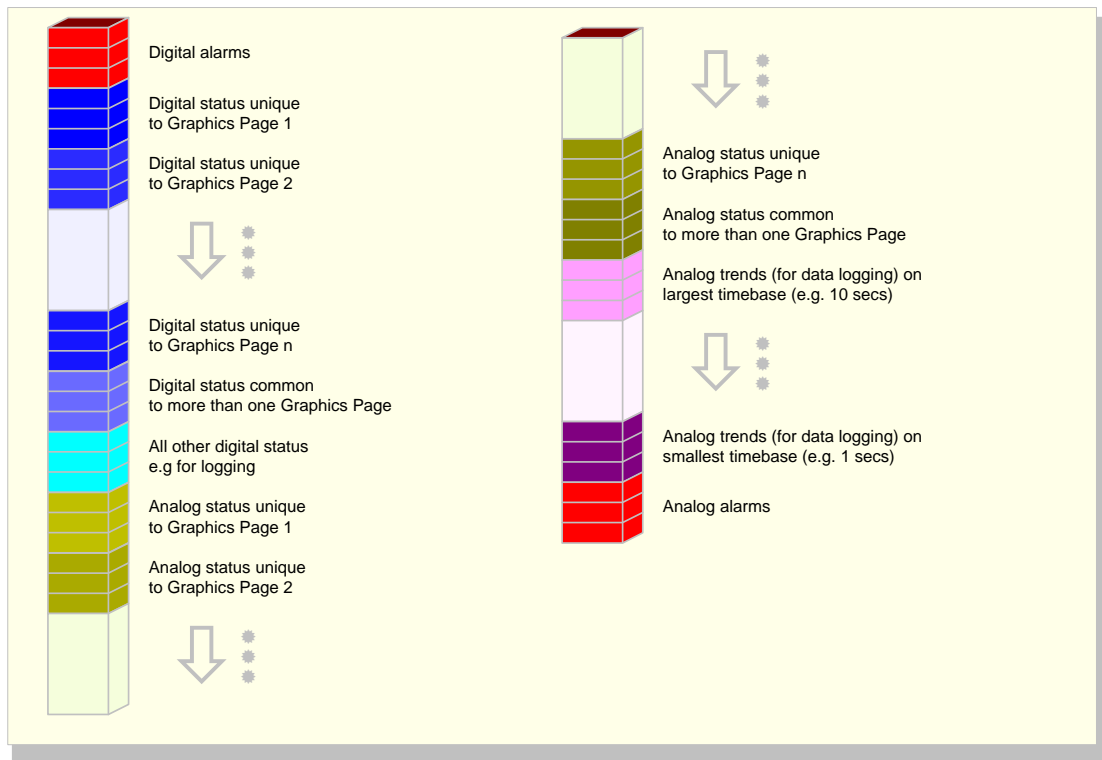
If you arrange (into groups) the registers that CitectHMI/SCADA will read, efficiency will increase.

CitectHMI/SCADA continually reads all registers associated with alarms. (If an alarm condition occurs, CitectHMI/SCADA can display the alarm immediately.) You should therefore group all registers that indicate alarm conditions.

Registers associated with status displays (objects, trends, etc.) are only read as they are required (i.e. when the associated graphics page is displayed) and are best grouped according to the pages on which they are displayed.

Registers used for data logging are read at a frequency that you define. They are best grouped according to the frequency at which they are read.

The following diagram shows an ideal register grouping for a CitectHMI/SCADA system.



While memory constraints and the existing PLC program may impose some limitations, grouping registers into discrete blocks (even if they are not consecutive blocks - as in the above diagram) will result in increased performance.

When you are designing your system, you should also allow several spare registers at the end of each block for future enhancements.

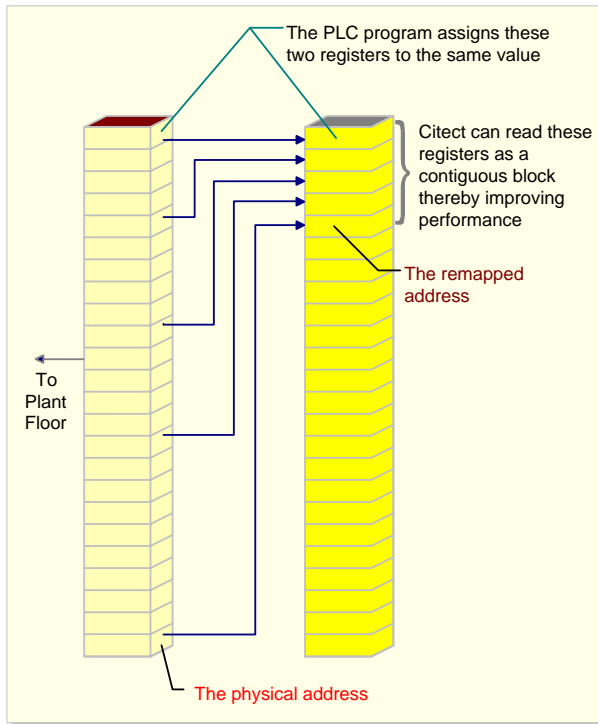
Remapping Variables in an I/O Device

Some PLCs allow you to remap (or copy) an I/O Device variable to another register address. You can use remapping in a CitectHMI/SCADA system for two purposes:

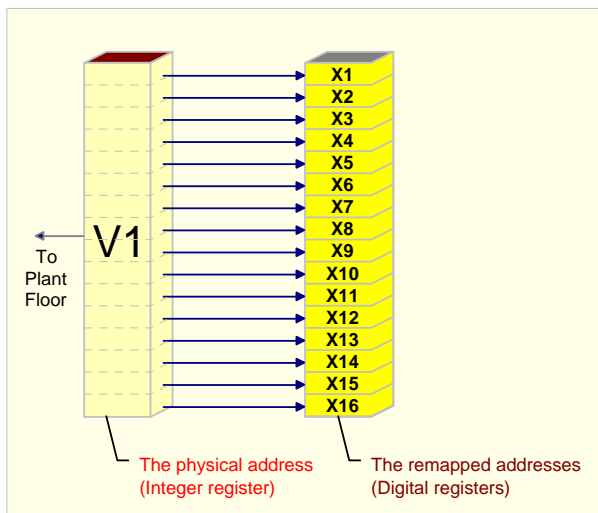
- To group registers more efficiently and therefore increase performance.
- To allow CitectHMI/SCADA to interpret a variable type (e.g. an analog variable) as a different variable type (e.g. a digital variable). For example, you can create additional digital addresses if an I/O Device has run out of digital addresses.

NOTE: Remapping of DISK and MEMORY I/O Devices is not supported.

To remap a variable in your PLC, you must design (or modify) the logic in the PLC to associate both addresses. CitectHMI/SCADA can then read or write the variable to and from the remapped address instead of the physical address.



You can also reassign one type of variable (e.g. an integer) to another type of variable (e.g. a digital variable).



To implement remapping in CitectHMI/SCADA, you should first create the variables in your project as you would normally. After you have created the variables, you can setup the remapping - specifying that any variable with an address in the desired range will be remapped. The I/O Server will re-direct the addresses at runtime as per the re-mapping instruction.

NOTE: Not all PLCs and/or CitectHMI/SCADA drivers support remapping. Citect does not recommend using remapping unless necessary. Please contact Citect Support if you need to evaluate whether the PLC and/or CitectHMI/SCADA driver support remapping.

Remapping Example:

Using the CCM protocol with a GE 9030 PLC, the following remapping could be used to optimise communication when reading some digitals. This example is based on the assumption of the PLC being setup correctly for remapping.

Variable Tags Database

The digitals are defined as follows.

Variable Tag Name	Motor_1_Running_Feedback
Data Type	Digital
I/O Device Name	Area1
Address	M1
Variable Tag Name	Motor_1_Fault
Data Type	Digital
I/O Device Name	Area1
Address	M3
Variable Tag Name	Motor_4_Running_Feedback
Data Type	Digital
I/O Device Name	Area1
Address	M4
.	
.	
.	
Variable Tag Name	Motor_4_Running_Feedback
Data Type	Digital
I/O Device Name	Area1
Address	M16

NOTE: Notice that the address M2 is not used. This will not cause any problems.


Remapping Database

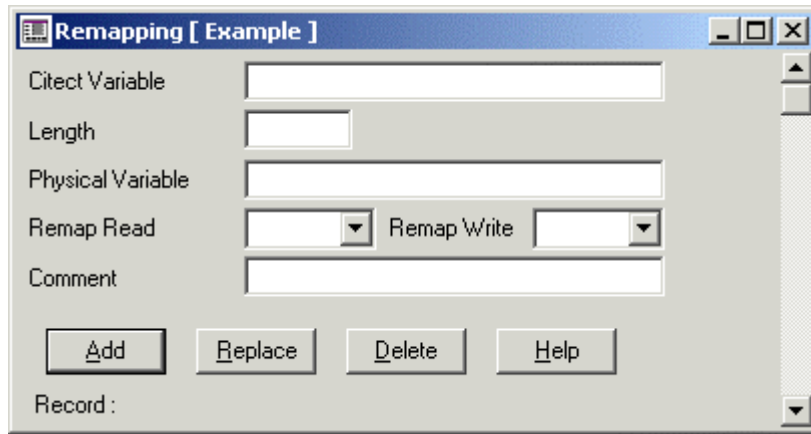
The remapping is defined as follows.

CitectHMI/SCADA Variable	Area1 M1
Length	16
Physical Variable	Area1 R10
Remap Read	True
Remap Write	False

The **Physical Variable** is an integer data type - it does not need to be defined in the Variable Tags database (but it can be).

➤ To remap a variable in CitectHMI/SCADA:

1. Select the Project Editor.  (or press this icon)
2. From the Communication menu select **Remapping**
3. Complete the properties in the **Remapping** form that appears. Use the **Help** button for more information about the fields.
4. Press the **Add** button to append a record you have created, or the **Replace** button if you have modified a record.



Remapping Properties

Remapping has the following properties:

CitectHMI/SCADA Variable

The first remapped (CitectHMI/SCADA) variable defined in the Variable Tags database (using the Tags form), for example: **Motor_1_Run**

Alternatively, you can use the direct **<Unit Name>|<Address>** format (using values specific to your I/O Device), for example: **IODev|X1|**

NOTE: The address entered here is remapped - at runtime the I/O Server will read/write data through the physical address instead.

Length

The number of remapped variables. CitectHMI/SCADA reads enough **Physical Variables** to remap this number of **CitectHMI/SCADA Variables**.

The Length must be less than the maximum request length of the protocol. The Protocol Overview displays the maximum request length of the protocol.

Physical Variable

The first physical variable in the PLC, for example: **ReMapIntV7**

This variable does **not** need to be defined in the Variable Tags database, and you can use the **<Unit Name>|<Address>** format (using values specific to your I/O Device), for example: **IODev|V7|**

Remap Read

Determines whether to perform the remapping for reads:

- 0 Read normal (not remapped) variables. The actual address of the **CitectHMI/SCADA Variables** will be read directly from the I/O Device, instead of through the **Physical Variable**. (You must use this mode if your I/O Device does not support remap reads.)
- 1 Read remapped variables(through the **Physical Variable**).

Remap Write

Determines whether to perform the remapping for writes:

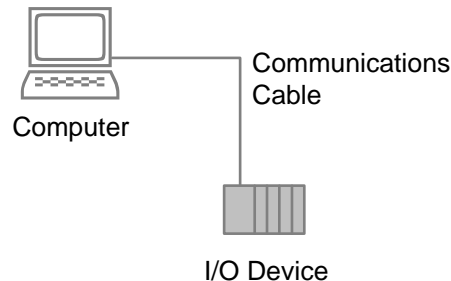
- 0 Write to normal (not remapped) variables. The actual address of the **CitectHMI/SCADA Variables** will be written directly in the I/O Device, instead of through the **Physical Variable**. (You must use this mode if your I/O Device does not support remap writes.)
- 1 Write to remapped variables (through the **Physical Variable**).

NOTE: To determine if your device supports remapped reads or writes, see the I/O Device Data Type help.

Serial Communications

Using a COM Port

The simplest CitectHMI/SCADA systems use a single computer connected to the I/O Device(s). You can connect an I/O Device directly to a communications port at the back of the computer with a standard RS-232 communications cable.



Special Options for the COMx Driver

Special Options (in the Ports form) are space separated and start with the dash (-) character immediately followed by the option characters. The following list covers most signal configurations:

DCD (Data Carrier Detect)

`-cATS0=1` Send the string 'ATS0=1<CR>' on startup to allow the modem to detect the baud rate the port is running at. Abandon transmit if DCD is low. Wait for incoming call to raise DCD.

DSR (Data Set Ready)

`-d` Data will be transmitted only when DSR is high.
`-di` Received data is ignored when DSR is low.
`-dMS` When transmitting a message the driver will wait up to 2000 ms for DSR to go high, then wait another MS milliseconds before transmitting.

CTS (Clear To Send)

`-h` Data will be transmitted only when CTS is high.
`-hMS` When transmitting a message the driver will wait up to 2000 ms for CTS to go high, then wait another MS milliseconds before transmitting.

RTS (Request To Send)

`-t` Driver will raise RTS only when transmitting.
`-ti` RTS is raised when there is enough room in the input buffer to receive incoming characters and drop RTS when there is not enough room in the input buffer.
`-to` RTS is raised only when there are characters to transmit.
`-tPRE,POST` When transmitting a message the driver will raise RTS for PRE milliseconds, transmit message, wait for POST milliseconds then drop RTS.

DTR (Data Terminal Ready)

`-r` Driver will raise DTR only when transmitting.
`-ri` DTR is raised when there is enough room in the input buffer to receive incoming characters and drop DTR when there is not enough room in the input buffer.
`-rPRE,POST` When transmitting a message the driver will raise DTR for PRE milliseconds,

transmit message, wait for POST milliseconds then drop DTR.

COMX Driver Special Options Reference

Special Options (in the Ports form) are space separated and start with the dash (-) character immediately followed by the option characters. Only a very small percentage of users will need to use the following options:

- cATS0=1** Send the string 'ATS0=1<CR>' on startup to allow the modem to detect the baud rate the port is running at. Abandon transmit if DCD is low. Wait for incoming call to raise DCD.

- d** Data will be transmitted only when DSR is high.

- di** Received data is ignored when DSR is low.

- dMS** When transmitting a message the driver will wait up to 2000 ms for DSR to go high, then wait another MS milliseconds before transmitting.

- e** Provides access to the output signal lines. The format is:
 "~EIAWXYZ" where **WXYZ** represents one of the following options:

SWF0	Simulate XOFF received
SWF1	Simulate XON received
RTS1	Set RTS high
RTS0	Set RTS low
DTR1	Set DTR high
DTR0	Set DTR low
BRK1	Set the device break line
BRK0	Clear the device break line

Example: "~EIADTR1" sets DTR high

- h** Data will be transmitted only when CTS is high.

- hMS** When transmitting a message the driver will wait up to 2000 ms for CTS to go high, then wait another MS milliseconds before transmitting.

- i** The string sent whenever the port is initialised. The tilde (~) and '\M' characters represent special instructions:
 ~: Delay for 500 milliseconds
 \M: Send carriage return
 Examples:
 ~Fred Wait 500 milliseconds and then send 'Fred'
 Fred\MMary Send 'Fred', a carriage return, and then 'Mary'

- nt** With some serial interfaces, line faults can cause the COMx read thread to

shutdown. If this happens, the driver does not recover after the fault. However, with the `-nt` (no terminate) option set, the thread is not shutdown, allowing the system to recover when the fault is rectified.

- nts** If errors occur when the COMx driver is starting up, it will not terminate, but will continue attempts to open the COMx port.
- r** Driver will raise DTR only when transmitting.
- ri** DTR is raised when there is enough room in the input buffer to receive incoming characters and drop DTR when there is not enough room in the input buffer.
- rPRE,POST** When transmitting a message the driver will raise DTR for PRE milliseconds, transmit message, wait for POST milliseconds then drop DTR.
- sc** Activates software flow control using XON and XOFF
XonLim: A number in bytes. This represents the level reached in the input buffer before the XON character is sent (30 bytes).
XoffLim: The maximum number of bytes accepted in the input buffer before the XOFF character is sent. This is calculated by subtracting (in bytes) 100 from the size of the input buffer.
- t** Driver will raise RTS only when transmitting.
- ti** RTS is raised when there is enough room in the input buffer to receive incoming characters and drop RTS when there is not enough room in the input buffer.
- to** RTS is raised only when there are characters to transmit.
- tPRE,POST** When transmitting a message the driver will raise RTS for PRE milliseconds, transmit message, wait for POST milliseconds then drop RTS.

➤ **How to set up CitectHMI/SCADA to use your computer's COM port:**

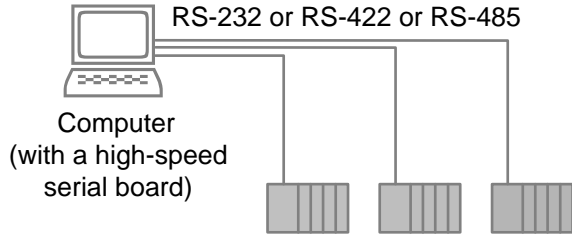
If you are using your computer's COM port, you do not need to install any additional software. To configure CitectHMI/SCADA you must:

- 1 Make sure that the **Boards** configuration has **COMx** as the **Type**, and the **Address** set to **0**. The **I/O Port**, **Interrupt** and **Special Opt** can all be left blank.
2. Enter the **Port Number** in the **Ports** configuration. The COM port number will usually be either **1** or **2**, and is set in the Ports section of the Control Panel. You can use the **Special Opt** field to modify the behaviour of the COMx driver - see below.

NOTE: You only need to define the COMx Board once. You can then add several Ports that use the same CitectHMI/SCADA board. For example, a COM port and two serial boards would be defined as one COMx board in CitectHMI/SCADA - with multiple ports.

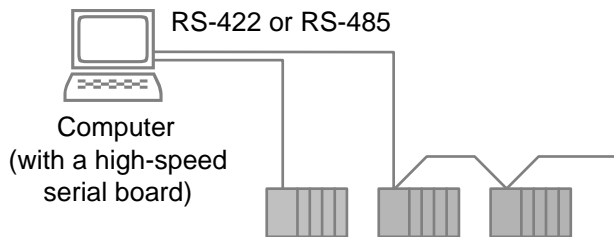
Using a Serial Board

The communications port on the computer is not designed for high speed communications, and reduces the performance of the system. For better performance, install a high speed serial board (such as a Digiboard) in your computer. High speed serial boards have several ports (usually 4, 8, or 16), so you can connect several I/O Devices to your CitectHMI/SCADA system.



You can use identical I/O Devices or I/O Devices supplied by different manufacturers. CitectHMI/SCADA supports all popular I/O Devices. You can connect any number of I/O Devices - the only limitation is the size of your computer. High speed serial boards are available for RS-232, RS-422, or RS-485 communication.

If you have several I/O Devices from the same manufacturer, and these I/O Devices support multi-drop communication, you can connect them to an RS-422 or RS-485 high speed serial board installed in your computer. (The RS-232 standard does not support multi-drop communication.)



Not all high speed serial boards support RS-422. You can use an RS-232/RS-422 or RS-232/RS-485 converter to achieve the same arrangement.



WARNING: Using a converter can introduce handshaking/timing problems.

Using Digiboards with Windows

CitectHMI/SCADA Version 5 uses the standard Digiboard drivers supplied by Digiboard with Windows. [CitectHMI/SCADA no longer supplies the drivers for these boards.](#)

The COM/Xi and MC/Xi are not supported for WIN95 or WINNT, but the PC/Xe and PC/Xi are.

Please note that you may use any other 3rd party serial board - you are not limited to using Digiboard serial boards.

➤ **How to set up CitectHMI/SCADA to use a serial board:**

NOTE: If you are using your computer's COM port, you do not need to install any additional software.

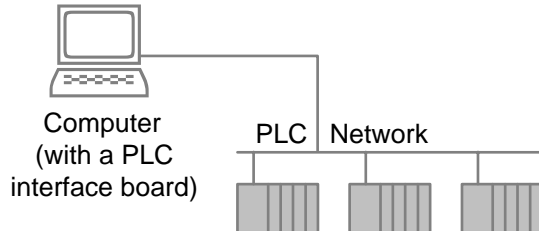
If you are using a serial board, to configure CitectHMI/SCADA you must:

1. Install the board in your computer and set it up under Windows as per the accompanying instructions. Make sure you are using the latest driver issued by the board manufacturer.
2. Make sure that the Boards configuration has COMx as the Type, and the Address set to 0. The I/O Port, Interrupt and Special Opt can all be left blank.
3. Enter the Port Number in the Ports configuration. The COM port number will usually be greater than 2, and is set in the Ports section of the Control Panel. You can use the Special Options field to modify the behaviour of the COMx driver.

NOTE: You only need to define the COMx Board once. You can then add several Ports that use the same CitectHMI/SCADA board. For example, a COM port and two serial boards would be defined as one COMx board in CitectHMI/SCADA - with multiple ports.

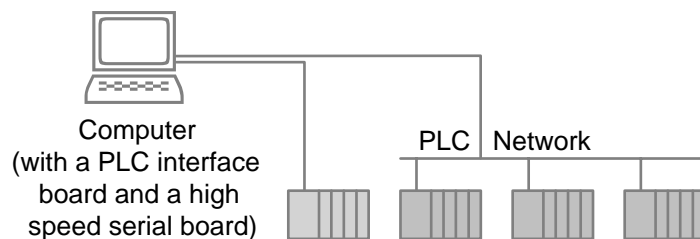
Using Proprietary Boards

In some cases (with some brands of PLCs), you can install a proprietary interface board in your computer. This PLC interface board is supplied by the PLC manufacturer, and you can connect it to a single PLC or to a PLC network. Usually, all interconnecting cables are supplied by the PLC manufacturer.



NOTE: With some PLCs, a high speed serial board provides better performance than a PLC interface board when the system is connected to more than one PLC.

You can mix both PLC interface boards and high speed serial boards in a single computer. You can, for example, connect a PLC network to a PLC interface board, and individual I/O Devices to a high speed serial board.

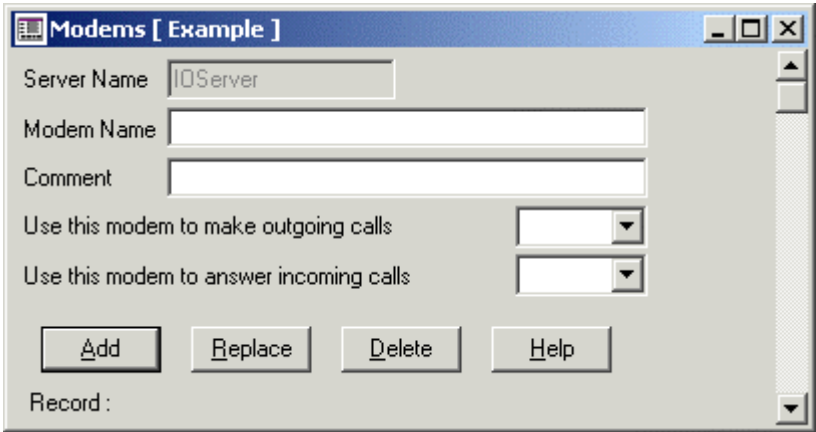


These examples are not exhaustive - there are many possible hardware arrangements for a CitectHMI/SCADA application. CitectHMI/SCADA is a flexible system - it imposes few restraints on the type (or manufacturer) of I/O Devices that you can use, or on the way you connect them to the computer.

➤ How to set up CitectHMI/SCADA to use a proprietary board:

If you are using a proprietary board (ie supplied by the PLC manufacturer), you must:

1. Install the board in your computer and set it up under Windows as per the accompanying instructions. Make sure you are using the latest driver issued by the PLC manufacturer.
2. If you are able, run diagnostics on the board before configuring CitectHMI/SCADA. This will enable you to see that the board works correctly.
3. Check that the I/O Port and Interrupt setting on the board are correct.
4. Configure the Boards and Ports as instructed by the PLC Specific help.



Serial Port Loop-Back Test

The serial port loop-back test can be used to test your serial hardware configuration. This test may be used with any COM port, whether it is local, or on a multi-port serial board (such as a Digiboard). The test can be performed internally or externally with loop-back cable attached.

Test Setup

1. **No other protocols should be configured.** You should temporarily delete any other boards and units while performing this test.

2. Configure a unit for each port to be tested. The **I/O Devices form** should look as follows:

Name: <unique name for the IO device>
Number: <unique network number for the IO device>
Address: NA
Protocol: LOOPBACK
Port Name: <the "Port Name" in the Ports form>

3. The following **Citect.ini** options are supported:

[LOOPBACK]

LoopBack = Set this to 1 if internal loop-back is to be performed (make sure this is deleted after running the test). When set to 0, a loop-back connector which ties pins 2 and 3 together is required at each port.

NOTE: The COMx driver does not support internal loop-back. The external loop-back is the default mode.

Size = Sets the maximum frame size. The length of each frame transmitted is random between 1 to 'Size'-1. The default size is 512.

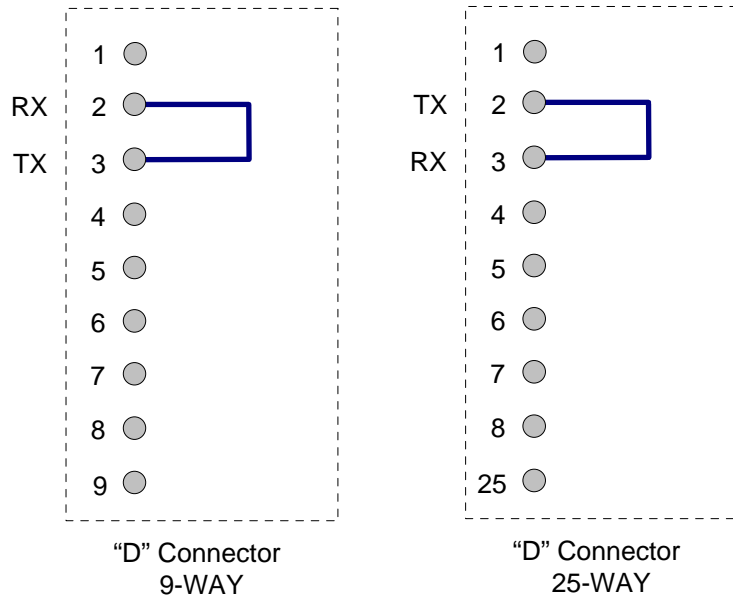
4. Start up CitectHMI/SCADA. Each port will transmit a frame of random length. This process is repeated when the entire frame is received..
5. Open the **kernel**, type "page driver" and press ENTER. Type "V" to set the display mode to 'verbose'. The following statistics will now be displayed:

Number of characters transmitted
 Number of frames transmitted
 Number of characters received
 Elapsed time in milliseconds
 Characters received per second
 Start time in milliseconds
 Total number of errors
 Error code of last error

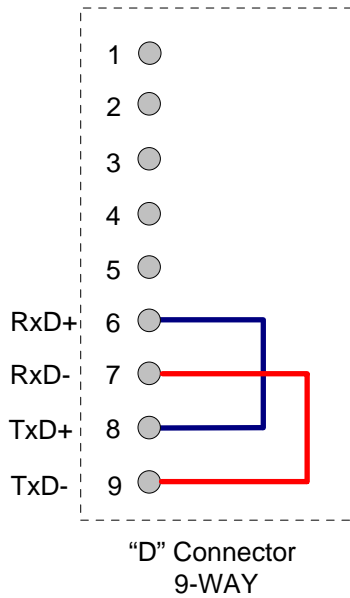
NOTE: The total number of errors should be 0. If the error is not zero there is a fault in your serial hardware.

Serial Port Loop-Back Cable

The following diagram shows the loop-back connections that you should use if using RS-232.



The following diagram shows the loop-back connections that you should use if using RS-422 or RS-485.



Setting Up Communications

➤ **To set up your I/O Device communication:**

1. Select the Project Editor.
(or press this icon) 
2. From the **Communication** menu select **Express Wizard**
- or -
1. Select the Citect Explorer
(or press this icon) 
2. Double click the **Express I/O Device Setup** icon in the **Communications** folder of the current project.
3. Follow the instructions given by the Wizard, using the buttons on the bottom for control.

NOTE: Each I/O Device/Protocol combination requires a unique setup for the Boards, Ports, and I/O Devices forms.
The CitectHMI/SCADA Online Help provides the information needed to set up each I/O Device. You can locate this information in the *Contents* under the "I/O Devices" book. You can also find all of the protocol details by looking under the "Protocols" book (this information is rarely needed). Once you are familiar with the help, use the *Index* to find I/O Device information quickly.

Manually Configuring Communications

In most cases, the Express Communications Wizard will be sufficient to set up your communications. If, however, your needs are very complex, you can set up communications manually by performing the following configuration steps within CitectHMI/SCADA:

1. Define an I/O Server in the I/O Server form. This defines the name of the CitectHMI/SCADA server that the I/O Device will communicate with.
2. Complete the Boards form. The Boards form defines which board (on your CitectHMI/SCADA computer) to use to communicate, such as the mother board, network card, serial board or a PLC communication card. Diappable remote I/O devices must use a COMX board.
3. Complete the Ports form. Often boards have multiple communication ports, and you must specify the port to use. Some modern equipment can have a number of logical (virtual) ports assigned to the one physical port. If using modems, you must specify a unique port name for each and -1 for the port number. Additionally you must specify the communication parameters (speed, etc.) and any special behaviour of that port.
4. Complete the I/O Devices form. This is where you define which I/O Device CitectHMI/SCADA is talking to, by specifying the address. The [protocol](#) is also defined at this level.
5. Run the Computer Setup Wizard to complete the configuration. This allows you to define your CitectHMI/SCADA computer as the I/O Server defined above. This is usually done after compilation of the project.
6. If you are using diappable remote I/O Devices, you will need to complete the Modems form. This is where you define how CitectHMI/SCADA will use a modem to communicate with remote I/O devices, eg. whether or not it will initiate and/or recieve calls.

NOTE: If there is no data to read or write, CitectHMI/SCADA will not communicate with an I/O Device - regardless of whether it is defined or not. You must create a variable tag, and use it somewhere in CitectHMI/SCADA, before CitectHMI/SCADA will do a read request. For example, use an integer variable to display a number on a page.

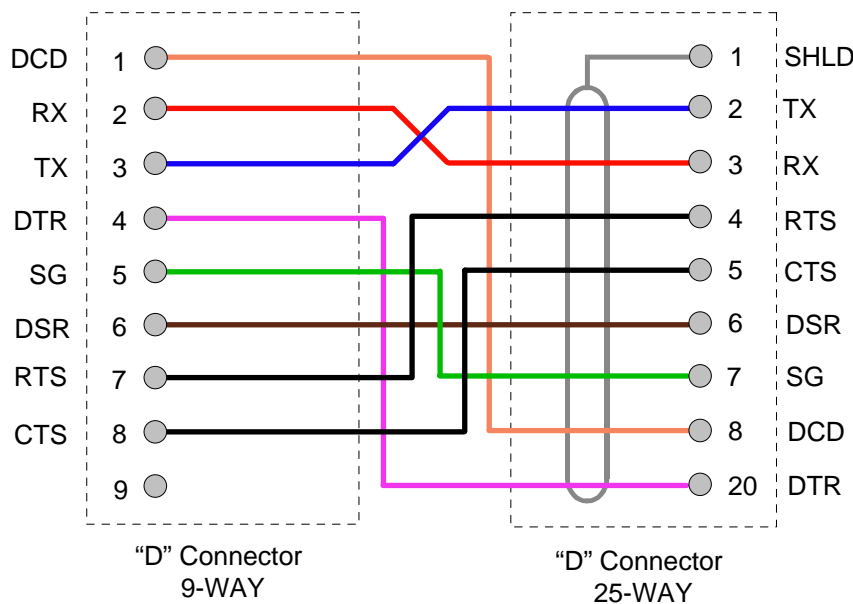
Wiring Cables

Creating and wiring data communication cables should be a routine task. This section contains pointers for additional insight into creating cables.

NOTE: If you are using proprietary hardware (ie a communication card installed in your CitectHMI/SCADA computer) you should always refer (exclusively) to the accompanying documentation. The hardware manufacturer should provide you with all the necessary wiring information (and often the communication cable).

Using a 9 to 25 pin Converter

In serial protocols help for an RS-232 wiring diagram, usually only the pin arrangement for a DB-25 (25 pin 'D' type) connector is given. In the case that you wish to use your Com port or a 9 pin serial card, a 9 to 25 diagram is usually supplied also. The diagram is as follows:



NOTE: The 9 and 25 pin connections above are considered standard. The 9 pin arrangement is common to most computer com ports.

The converter is 'straight through'. This means that pins that are labelled the same are connected together (ie TX goes to TX).

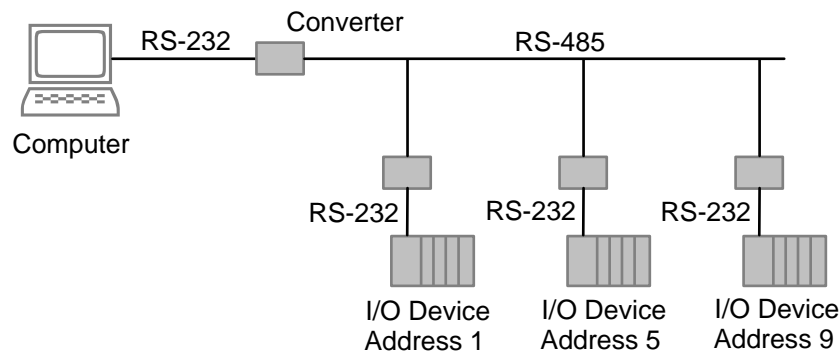
The diagram is intended to show you how to wire your serial cable to a 9 pin port. You do not need to create a 25 pin connection. Consider the following scenario:

- You want to connect to your computer's com port (ideal for a small system) using RS-232.
- The wiring diagram given in the help shows only a 25 pin connection.
- Create the cable as normal, but instead of using the 25 pin connector use a 9 pin. Instead of joining the wires to the pins on the 25 pin diagram, use the above diagram to see which pins to use on the 9 pin. eg instead of using pin 2 on the 25 pin (TX) use pin 3 on the 9 pin (TX).

Using an RS232/485 Converter

Using an RS232/485 converter is quite common. It presents a cheaper alternative for those wishing to use RS-485 but don't want to bother with an RS-422/485 serial board for the computer. RS-485 has significant advantages over RS-232, such as longer distances, faster transmission, noise immunity. Even more significant is that RS-232 does not support multi-drop.

The following diagram shows how converters may be used to allow a configuration that would not be achievable with RS-232.



This arrangement is only available if the protocol supports multiple I/O Devices.. RS-422 can be used also if supported by the protocol. Obviously the maximum data transfer rate would be less than that of a high speed serial board.

NOTE: No details of wiring can be given here as it will vary from manufacturer to manufacturer. Though generally the devices are defined as DCE and should be wired as such.

DTE and DCE

Some people are often confused by DTE and DCE. They are defined as follows:

DTE or Data Terminal Equipment. This represents the equipment that ultimately acts as a data source or data sink (ie to further process the data). Computers and PLCs are usually regarded as DTE.

DCE or Data Communications Equipment. This represents a device that transmits data between a DTE and a physical communications link. Usually responsible for establishing and maintaining a data transmission connection. Normally DCE refers to a modem.

Often it will be necessary to use DCE equipment in your control communications. These devices should be very simple to wire as the only difference DCE and DTE is the use of the TX (transmit) and RX (receive) pins. Except where stated otherwise, all wiring diagrams in CitectHMI/SCADA help are for DTE. To use the same cable for DCE simply reverse the TX and RX connections, or follow the following rule of thumb:

- DTE to DCE: Join DCE-RX to DTE-RX and DCE-TX to DTE-TX
- DTE to DTE: Join DCE-RX to DTE-TX and DCE-TX to DCE -RX

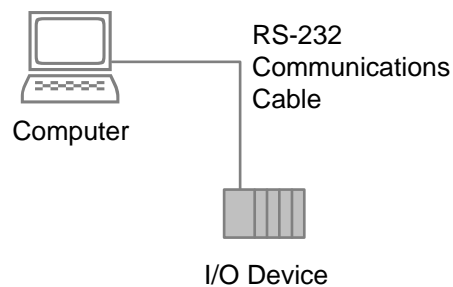
Common Serial Communication Standards

There are many data communication standards, a number of them (described here) are considered common. This section aims to outline the basic functionality and features of these common standards; RS-232C, RS-422, RS-485.

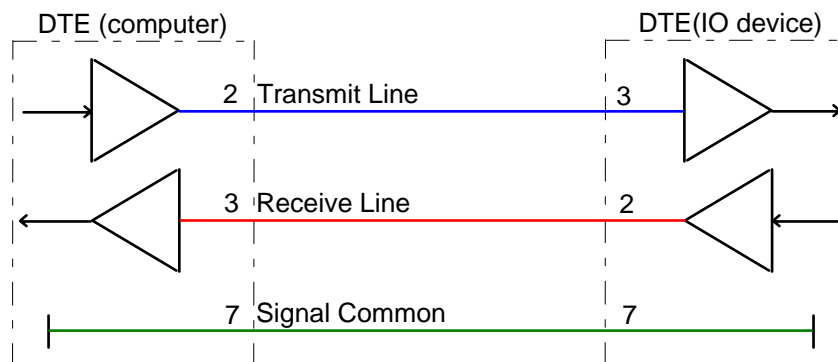
RS-232C (or EIA-232C or RS-232)

RS-232C is the most common serial data communication interface standard. This standard defines the electrical and mechanical details of the interface but **does not define a protocol**. The standard covers the electrical signal characteristics, the mechanical interface characteristics (pin out etc) and functional description of control signals etc.

- Point-to-point communication. Between only 2 devices.



- Communication is full-duplex. A single wire for each direction and a ground wire. This means that generally only 3 wires need to be connected for most applications. The diagram below shows the 'standard' pins for a DB-25 Connector.

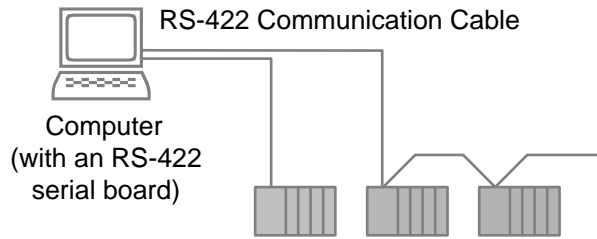


- Less than 75m maximum length at 19.2K maximum Baud rate. But up to 900 meters may be achievable at 900 Baud.

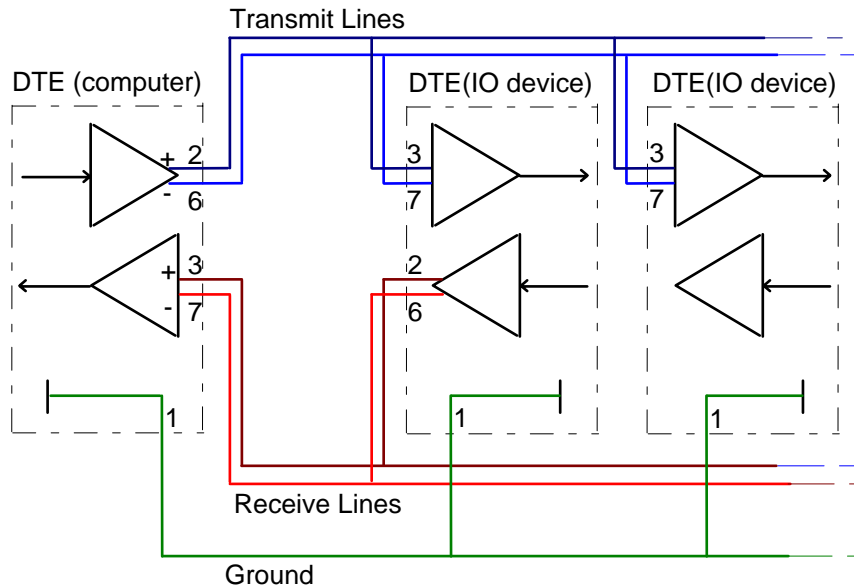
RS-422 (or EIA-422)

RS-422 is recommended as it has significant benefits over RS-232C. This standard covers the electrical signal characteristics and functional description of control signals only. **It does not define the protocol**, but the protocol used should support multiple unit addressing to fully utilise this standard.

- Uses differential signals (difference between to line voltages) which provide greater noise immunity.
- Limited multi-drop communication. This means that there may be multiple receivers (**but only 1 transmitter**) on each line.



- Communication between two devices is full-duplex. Two wires are used for each direction and also one ground wire. This means that generally only 5 wires need to be connected for most applications. The diagram below shows a commonly used pin arrangement for a DB-9 Connector.



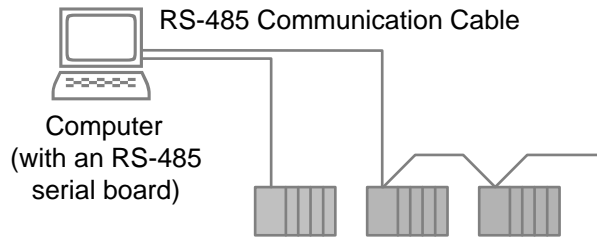
NOTE: Only one transmitter is allowed per line though there may be multiple receivers. This means only two devices may have full-duplex (half-duplex for 2 wire) while the other devices have only simplex communication. This is reflected in the above diagram.

- Distances up to 1200m and transfer rates up to 10Mbps are achievable.
- The protocol used with this standard must take care of who (i.e. which device) is allowed to transmit at any one time. This allows each device to act as a transmitter when requested.

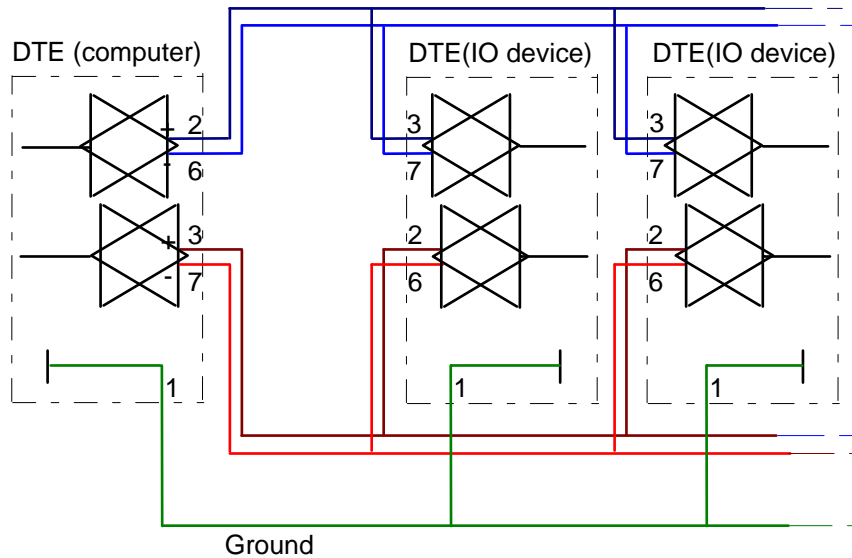
RS-485 (or EIA-485)

RS-485 is an improved version of RS-422. This standard covers the electrical signal characteristics and functional description of control signals only. **It does not define the protocol**, but the protocol used should support multiple unit addressing and bus contention to fully utilise this standard. The major advantage is that all devices can transmit and receive on the same line.

- Electrically very similar to 422. Logic levels, transfer rates and maximum distance are almost identical.
- RS-485 supports multiple **transmitters and receivers** on **each** line. This is the improvement on RS-422.



- Communication may be either half-duplex or full-duplex. Two wires are used for each direction and also one ground wire. This means that only three wires need to be connected for most half-duplex applications. Five wires are needed for most full-duplex applications. The diagram below shows a commonly used pin arrangement for a DB-9 Connector.



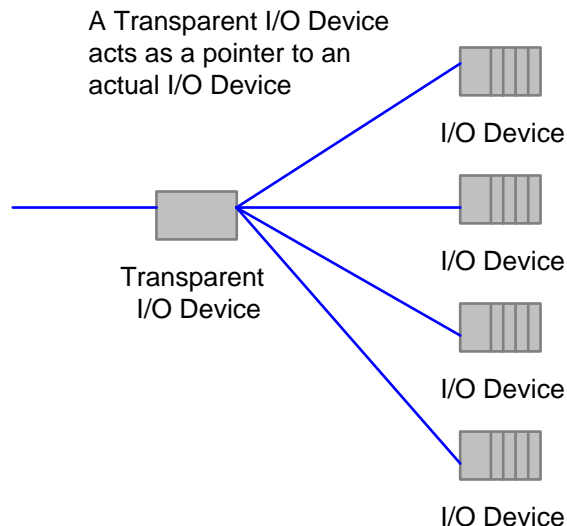
NOTE: RS-485 supports both full and half-duplex. The above diagram shows a full-duplex arrangement. Unlike RS-422 each IO device is able to transmit and receive on each line. If the arrangement were half-duplex, only one pair of transmission lines would be needed (rather than two pairs shown above).

- The protocol used with this standard must take care of who (ie which device) is allowed to transmit at any one time. This allows each device to act as a transmitter when requested.

Using a Transparent I/O Device

If you have several identical I/O Devices (e.g. controlling identical processes), you can simplify your configuration with a Transparent I/O Device. A Transparent I/O Device operates like a pointer to an actual I/O Device, and allows you to reassign reads and writes to the actual I/O Device.

A Transparent I/O Device can only be used under special conditions, and you should only use a transparent I/O Device if you understand exactly how it works. If your project suits a transparent I/O Device, this feature can reduce configuration considerably.



In addition to the I/O Device configuration (required to establish communication), you must also setup the Transparent I/O Device. The Transparent I/O Device requires no board or port configuration. In the I/O Device form, the transparent I/O Device must have the same I/O Device name as the physical I/O Devices that are referenced by the transparent I/O Device, the address may be left blank and "TRANSPARENT" must be entered into the port name field.

You can then configure one page, (for example called "TransPg") and define a number of buttons (on a menu page) that each call the IODeviceControl() function followed by PageDisplay("TransPg"). For example, to show the data from the physical I/O Device No 1 you would use the following command:

```
IODeviceControl(6, 2, 1); PageDisplay("TransPg");
```

and to display data for the physical I/O Device No 2,

```
IODeviceControl(6, 2, 2); PageDisplay("TransPg");
```

Alternatively, you can use the following Cicode function:

```
Function
ShowUnit(INT UnitNo)
    IODeviceControl(6, 2, UnitNo);
    PageDisplay("TransPg");
End
```

The first button would then call ShowUnit(1), the second button ShowUnit(2) and so on.

NOTE: In the above example the transparent device is I/O Device No 6.

➤ **To define a Transparent I/O Device:**

1. Configure a new I/O Device

2. Enter a unique number for the I/O Device
3. Enter TRANSPARENT for the Port Name.

➤ **To reassign a Transparent I/O Device to an actual I/O Device:**

1. Use the IODeviceControl function.

NOTES: 1) You can only use this feature if all I/O Devices that the Transparent I/O Device references are identical in every respect.

2) Variables read from a transparent I/O Device are not compatible with Super Genies or with the advanced DDE features.

3) The number of points in a transparent I/O Device is counted as follows:
The number of points in the I/O Device x the number of physical I/O Devices you could reassign to. For example, if you have 20 points in a transparent I/O Device, and 10 physical I/O Devices of the same protocol, the point count is $20 \times 10 = 200$ Points - you can read up to 200 physical points from the plant. If you are already reading these points, they are not counted twice.

Citect Driver Accreditation

Each driver-I/O Device combination supported by CitectHMI/SCADA is subject to the Citect CitectHMI/SCADA Driver Quality and Accreditation System (CiTDriversQA96). Comprised of a quality process and its standards, this system promotes extremely high reliability, robust design and operation, ease of support, and efficient performance.

Because drivers can be written by any third-party developer, not all CitectHMI/SCADA drivers undergo the same quality control procedures. To enable users to distinguish between drivers of different standards, the following categories are used:

- [Accredited](#) - Level 1;
- [Accredited](#) - Level 2; or
- Functionally Stable.

NOTE: The help for every I/O Device indicates which category the driver-I/O Device combination belongs in.

Advanced Driver Information

Variable (Digital) Limitations

Devices often have memory areas that are of a designated data type, like Byte, Integer, or Word. Some protocols do not support the reading and writing of data in these memory areas using a different data type. This situation is most common in the case of reading and writing of individual bits within the data types like Bytes, Integers, and Words. This is probably because the original designer of the protocol did not anticipate that the user would want to access these data types in a format other than the designated type.

In this case, reading individual bits within these larger data types is done by simply reading the designated data type and getting the CitectHMI/SCADA Driver to break it down into individual bits. Writing to bits within the larger data types is more complicated, as writing to one bit within the larger data type will at the same time overwrite the other bits within that same data type. To overcome this limitation, a 'read-modify-write' scenario can be used to write to a bit within the larger data type. Using this approach, the CitectHMI/SCADA driver will read the larger data type, modify the appropriate bit within the larger data type, and then write the larger data type back to the device.

This 'read-modify-write' method has a serious operational concern that the user should be aware of: if the device modifies the larger data type after the CitectHMI/SCADA driver has read it, but before CitectHMI/SCADA has written the new value, then any changes made by the device will be overwritten and lost. This issue could be serious in a control system, and it is recommended that the device and CitectHMI/SCADA be configured so that only one of these systems writes to the data types of this kind.

Consider the following example:

1. The initial state of a PLC register is 0x02h.
2. The CitectHMI/SCADA driver reads the value of this register (effectively making a copy) in preparation for a change to bit 3.
3. However, before the CitectHMI/SCADA driver writes its change back to the PLC, the PLC code changes the value of bits 0 and 4 of this register to 0x13h.
4. The CitectHMI/SCADA driver then changes bit 3 of its copy of the register to 0x0Ah. When it writes to the PLC, it overwrites the PLC's copy of the whole register (not just the changed bit). Because the PLC code modified bits 0 and 4 in the interval between CitectHMI/SCADA's read and write, these changes are overwritten.

Generic Driver Errors

The following errors are generic to all CitectHMI/SCADA drivers. A driver error must be mapped to a Generic Error before CitectHMI/SCADA can interpret it.

GENERIC_ADDRESS_RANGE_ERROR (0x0001 | SEVERITY_ERROR)

A request was made to a device address that does not exist. For example, an attempt was made to read register number 4000 when there is only 200 registers in the device.

GENERIC_CMD_CANCELED (0x0002 | SEVERITY_ERROR)

The server cancelled the command while the driver was processing it. This may happen if the driver is taking too long to process the command. Check the timeout and retries for the driver.

GENERIC_INVALID_DATA_TYPE (0x0003 | SEVERITY_ERROR)

A request was made specifying a data type that is not supported by the protocol. This error should not occur during normal operation.

GENERIC_INVALID_DATA_FORMAT (0x0004 | SEVERITY_ERROR)

A request contains invalid data, eg. writing to a floating-point address with an invalid floating-point number. Check the CitectHMI/SCADA database.

GENERIC_INVALID_COMMAND (0x0005 | SEVERITY_ERROR)

The server sent a command to the driver that it did not recognise. This error should not occur during normal operation.

GENERIC_INVALID_RESPONSE (0x0006 | SEVERITY_ERROR)

There is a problem with the communication channel causing errors in the transmitted data.

GENERIC_UNIT_TIMEOUT (0x0007 | SEVERITY_ERROR)

A device is not responding to read or write requests. The driver sent a command to the device and the device did not respond within the timeout period.

GENERIC_GENERAL_ERROR (0x0008 | SEVERITY_ERROR)

Unmapped driver specific errors are normally reported as a general error. Refer to the protocol-specific errors listed with the protocol you are using.

GENERIC_WRITE_PROTECT (0x0009 | SEVERITY_ERROR)

A write operation was attempted to a location that is protected against unauthorised modification. Change the access rights to this location to permit a write operation.

GENERIC_HARDWARE_ERROR (0x000A | SEVERITY_UNRECOVERABLE)

A problem exists with either the communication channel, server or device hardware. Examine all hardware components. The server's operation may no longer be reliable.

GENERIC_UNIT_WARNING (0x000B | SEVERITY_WARNING)

The communication link between the server and the device is functioning correctly, however the device has some warning condition active, for example, the device is in program mode.

GENERIC_UNIT_OFFLINE (0x000C | SEVERITY_SEVERE)

The device is in off-line mode, preventing any external communication. This error will cause any stand-by units to become active. CitectHMI/SCADA will attempt to re-initialise the unit.

GENERIC_SOFTWARE_ERROR (0x000D | SEVERITY_SEVERE)

An internal software error has occurred in the driver. This error should not occur during normal operation.

GENERIC_ACCESS_VOILATION (0x000E | SEVERITY_ERROR)

An attempt has been made by an unauthorised user to access information. Check the user's access rights.

GENERIC_NO_MEMORY (0x000F | SEVERITY_UNRECOVERABLE)

The server or driver has run out of memory and cannot continue execution. Minimise buffer and queue allocation or expand the server computer's memory (physical or virtual memory).

GENERIC_NO_BUFFERS (0x0010 | SEVERITY_ERROR)

There are no communication buffers left to allocate. The performance of the server may be

reduced, however it can still continue to run. Increase the number of communication buffers.

GENERIC_LOW_BUFFERS (0x0011| SEVERITY_WARNING)

This error may occur in periods of high transient loading with no ill effects. If this error occurs frequently, increase the number of communication buffers.

GENERIC_TOO_MANY_COMMANDS (0x0012| SEVERITY_WARNING)

Too many commands have been sent to the driver. If you are using a NETBIOS driver, increase the number of NETBIOS control blocks.

GENERIC_DRIVER_TIMEOUT (0x0013 | SEVERITY_ERROR)

The server is not receiving any response from the driver. This error should not occur during normal operation.

GENERIC_NO_MORE_CHANNELS (0x0014 | SEVERITY_SEVERE)

Each driver can only support a fixed number of communication channels. You have exceeded the limit. The command or data request has not been completed.

GENERIC_CHANNEL_OFFLINE (0x0015 | SEVERITY_SEVERE)

A communication channel is currently off-line, disabling communication. The server cannot initialise the communication channel or the channel went off-line while running. All devices (units) connected using this channel will be considered to be off-line so this will cause any stand-by devices to become active. CitectHMI/SCADA will attempt to re-initialise the channel.

GENERIC_BAD_CHANNEL (0x0016| SEVERITY_SEVERE)

The server has attempted to communicate using a channel that is not open.

GENERIC_CHANNEL_NOT_INIT (0x0017 | SEVERITY_SEVERE)

The server is attempting to communicate with a channel that has not been initialised. This error should not occur during normal operation. The command or data request has not been completed. If the problem persists, contact Citect Support.

GENERIC_TOO_MANY_UNITS (0x0018 | SEVERITY_SEVERE)

A channel has too many devices attached to it. This error should not occur during normal operation.

GENERIC_INVALID_DATA (0x0019 | SEVERITY_ERROR)

The data requested is not in a valid format or expected type.

GENERIC_CANNOT_CANCEL (0x001A | SEVERITY_WARNING)

The server tried to cancel a command, but the driver could not find the command. This error should not occur during normal operation.

GENERIC_STANDBY_ACTIVE (0x001B | SEVERITY_WARNING)

Communication has been switched from the primary to the stand-by unit(s). The server returns this message when a hot changeover has occurred.

GENERIC_MSG_OVERRUN (0x001C | SEVERITY_ERROR)

A response was longer than the response buffer. If this error occurs on serial communication drivers, garbled characters may be received. Check the communication link and the baud rate of

the driver.

GENERIC_BAD_PARAMETER (0x001D | SEVERITY_ERROR)

There is a configuration error, e.g. invalid special options have been set.

GENERIC_STANDBY_ERROR (0x001E | SEVERITY_WARNING)

There is an error in a stand-by unit.

GENERIC_NO_RESPONSE (0x001F | SEVERITY_ERROR)

There is no response from the communications server.

Standard Driver Errors

The following errors are low-level errors which are generic to all CitectHMI/SCADA drivers. These errors are all mapped to Generic errors so that CitectHMI/SCADA can recognise them. Most drivers also have a set of driver specific errors in addition to these errors.

0 (0x00000000) **NO_ERROR**

No error condition exists.

1 (0x00000001) **DRIVER_CHAR_OVERRUN**

Transmitted characters could not be received fast enough. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

2 (0x00000002) **DRIVER_CHAR_PARITY**

Parity error in received characters. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

3 (0x00000003) **DRIVER_CHAR_BREAK**

A break was detected in the receive line. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

4 (0x00000004) **DRIVER_CHAR_FRAMING**

Framing error. Check the baud rate. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

5 (0x00000005) **DRIVER_MSG_OVERRUN**

The message received from the device was too long. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

6 (0x00000006) **DRIVER_BAD_CRC**

The checksum in the received message does not match the calculated value. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

7 (0x00000007) **DRIVER_NO_STX**

The start of text character is not present. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

8 (0x00000008) **DRIVER_NO_ETX**

The end of text character is not present. This error is mapped to Generic Error
GENERIC_INVALID_RESPONSE.

9 (0x00000009) **DRIVER_NOT_INIT**

The driver has not been initialised. This error is mapped to Generic Error
GENERIC_UNIT_OFFLINE.

10 (0x0000000A) **DRIVER_BAD_TRANSMIT**

Cannot transmit message. This error is mapped to Generic Error GENERIC_UNIT_OFFLINE.

11 (0X0000000B) **DRIVER_CANNOT_RESET**

Cannot reset serial driver. This error is mapped to Generic Error GENERIC_CHANNEL_OFFLINE.

12 (0X0000000C) **DRIVER_BAD_LENGTH**

Response length is incorrect. This error is mapped to Generic Error
GENERIC_GENERAL_ERROR.

13 (0X0000000D) **DRIVER_MSG_UNDERRUN**

Message length too short. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

15 (0X0000000F) **DRIVER_INVALID_COMMAND**

The command from the server is invalid. This error is mapped to Generic Error
GENERIC_INVALID_COMMAND.

16 (0X00000010) **DRIVER_NO_TIMER**

Cannot allocate timer resource for the driver. This error is mapped to Generic Error
GENERIC_HARDWARE_ERROR.

17 (0x00000011) **DRIVER_NO_MORE_CHANNELS**

Too many channels specified for device. This error is mapped to Generic Error
GENERIC_NO_MORE_CHANNELS.

18 (0x00000012) **DRIVER_BAD_CHANNEL**

The channel number from the server is not opened. This error is mapped to Generic Error
GENERIC_BAD_CHANNEL.

19 (0x00000013) **DRIVER_CANNOT_CANCEL**

Command cannot be cancelled. This error is mapped to Generic Error
GENERIC_CANNOT_CANCEL.

20 (0x00000014) **DRIVER_CHANNEL_OFFLINE**

The channel is not online. This error is mapped to Generic Error GENERIC_CHANNEL_OFFLINE.

21 (0x00000015) **DRIVER_TIMEOUT**

No response have been received within the user configure time. This error is mapped to Generic
Error GENERIC_UNIT_TIMEOUT.

22 (0x00000016) **DRIVER_BAD_UNIT**

The unit number from the server is not active or is out of range. This error is mapped to Generic
Error GENERIC_UNIT_OFFLINE.

23 (0x00000017) **DRIVER_UNIT_OFFLINE**

The unit is not online. This error is mapped to Generic Error GENERIC_UNIT_OFFLINE.

24 (0x00000018) **DRIVER_BAD_DATA_TYPE**

The data type from the server is unknown to the driver. This error is mapped to Generic Error GENERIC_INVALID_DATA_TYPE.

25 (0x00000019) **DRIVER_BAD_UNIT_TYPE**

The unit type from the server is unknown to the driver. This error is mapped to Generic Error GENERIC_INVALID_DATA_TYPE.

26 (0x0000001A) **DRIVER_TOO_MANY_UNITS**

Too many units specified for channel. This error is mapped to Generic Error GENERIC_TOO_MANY_UNITS.

27 (0x0000001B) **DRIVER_TOO_MANY_COMMANDS**

Too many commands have been issued to the driver. This error is mapped to Generic Error GENERIC_TOO_MANY_COMMANDS.

29 (0x0000001D) **DRIVER_CMD_CANCELED**

Command is cancelled. This error is mapped to Generic Error GENERIC_COMMAND_CANCELLED.

30 (0x0000001E) **DRIVER_ADDRESS_RANGE_ERROR**

The address/length is out of range. This error is mapped to Generic Error GENERIC_ADDRESS_RANGE_ERROR.

31 (0x0000001F) **DRIVER_DATA_LENGTH_ERROR**

The data length from the server is wrong. This error is mapped to Generic Error GENERIC_INVALID_RESPONSE.

32 (0x00000020) **DRIVER_BAD_DATA**

Cannot read the data from the device. This error is mapped to Generic Error GENERIC_INVALID_DATA.

33 (0x00000021) **DRIVER_DEVICE_NOT_EXIST**

Device specified does not exist. This error is mapped to Generic Error GENERIC_HARDWARE_ERROR.

34 (0x00000022) **DRIVER_DEVICE_NO_INTERRUPT**

Device specified does not support interrupt. This error is mapped to Generic Error GENERIC_HARDWARE_ERROR.

35 (0x00000023) **DRIVER_BAD_SPECIAL**

Invalid special options in port database. This error is mapped to Generic Error GENERIC_BAD_PARAMETER.

36 (0x00000024) **DRIVER_CANNOT_WRITE**

Cannot write to variable. This error is mapped to Generic Error GENERIC_GENERAL_ERROR.

37 (0x00000025) **DRIVER_NO_MEMORY**

The driver has run out of memory and cannot continue execution. Minimise buffer and queue

allocation or expand the computer's memory (physical or virtual memory). This error is mapped to Generic Error GENERIC_NO_MEMORY.