CHURCHILL CONTROLS
www.churchill-controls.co.uk

Micro_Link

APPLICATION NOTE AN015

Using the Bus_Link Port

Summary

In addition to transferring 'Real-World' data, *Micro_Link* can also interface directly to a PLC, SCADA system or third party telemetry system over the *Bus_Link* serial port. This document outlines the supported options for accessing this information. More detailed information can be found in the *Data_Link 2000 Technical Manual*.

This document assumes that the user has already established the communications network from Micro_Link modules (possible also integrating Nano_Link outstations or repeaters).

For the sake of simplicity this document assumes each item of third party equipment is a PLC. However, it could equally be a SCADA system, a regional telemetry outstation or an intelligent instrument.

1 Overview

1.1 Introduction

Many PLCs have the ability to communicate with other devices via a serial link. This is generally done at one of two electrical interface standards – RS232 or RS485. The PLC manufacturer also defines the protocol (language) used for communication. The device to which it is linked, along with the data rate and format must match this.

Virtually all PLC protocols operate on a master/slave basis, whereby a master device can ‘talk’ to one or more slaves. All communication is instigated by the master, and each slave responds only to the commands addressed to it. Many PLCs operate only in slave mode. Numerous SCADA systems have been developed to communicate with PLCs, so these generally implement only the relevant master mode. Many regional telemetry outstations have the facility to link to PLCs, and they generally implement only master mode.

Micro_Link has been designed to interface to any of these systems. A serial interface port is included within each ***Micro_Link***, and is named ***Bus_Link*** to signify that it is a general-purpose port for communication with various PLC busses. It can be configured to emulate either master or slave in any required protocol, data rate and format, and can use either RS232 or RS485 signal levels.

1.2 Configuration Process

The process to set up ***Bus_Link*** communications within ***Micro_Link*** follows the following sequence:

1.2.1 Configure the Communications Network

Every network comprises a base-station and one or more outstations. Each outstation can be configured to also act as a repeater if necessary to access more distant outstations.

1.2.2 Decide whether the Micro_Link is to be configured as a master or a slave

Overall system configuration is far simpler if the ***Micro_Link*** is configured as the master. However, if the PLC can only implement master protocol the ***Micro_Link*** can be configured as a slave.

1.2.3 Establish a hardware link between each PLC and Micro_Link

The wiring between the two could use either RS232 or RS485. The user needs to connect the items together and prove that the link is functioning. This is most easily done by installing a simple configuration, then using DCD Diagnostics to prove that communication is working.

1.2.4 Configure Micro_Link and/or the PLC to pass the required data

Once communications are functioning the user can define the configuration necessary to pass the required data.

The rest of this document expands on the process.

2 Protocol and Interface

2.1 Protocols

Various PLC types are supported by ***Micro_Link***, as it can communicate in a number of different languages (protocols). The most commonly implemented is Modbus, as the majority of PLC and SCADA systems support Modbus in one form or another – either inherently, or by the addition of a Modbus card. However should the user not wish to, or be able to, talk Modbus, a number of other PLC manufacturers' own protocols are also supported.

2.1.1 Modbus

Modbus is an open protocol based on master/slave architecture. It is popular, well established, relatively easy to implement and reliable.

Since it is so easy to implement, Modbus has gained wide market acceptance wherever automation or management systems (BMS) need to communicate with other devices, and is probably the most widely implemented automation protocol.

The simplicity of Modbus RTU messages is a mixed blessing. On the one hand, the simple message structure ensures widespread, rapid and accurate implementation, but on the other hand, various companies have corrupted the basic 16-bit Modbus RTU register structure to pack in floating point, queues, ASCII text, tables and other types of non-Modbus data.

Micro_Link is fully configurable and allow user adjustments to compensate for the above issues

Modbus comes in two varieties – RTU and ASCII – both of which are supported by Micro_Link.

The Modbus RTU protocol is a fast, efficient binary protocol that uses all 8 bits of each character. The Modbus ASCII protocol uses two ASCII characters to send and receive each character of the Modbus message. While less efficient than its RTU counterpart, it is also widely accepted as a communication standard for remote telemetry applications.

2.1.2 Mitsubishi

Mitsubishi PLC's can be configured to either 'dedicated protocol' mode, or 'no-protocol' mode.

In 'no-protocol' mode, the PLC programmer must define the protocol in ladder logic, which is an onerous task.

Micro_Link therefore supports the dedicated protocol, which is the 'Melsec-A' protocol.

Both 'FX' and non-'FX' types of Mitsubishi PLC are supported using this protocol, with the configuration requirements for each being detailed in the Data_Link 200 Technical Manual.

Note that some Mitsubishi serial interface modules do not support dedicated protocol..

2.1.3 Allen-Bradley

Micro_Link supports the standard Allen Bradley protocol 'DF1'.

Note that, although the protocol may be supported on various PLCs, ***Micro_Link*** will only connect over a serial link.

This means it will not connect over any network based link, such as DH, DH+, DH485 or ControlNet, unless a suitable RS232 module is fitted in the network.

Also, ***Micro_Link*** will only support the Half-Duplex method of transmission. This is however sufficient to connect up to 255 nodes simultaneously, when using the RS485 connection option on ***Micro_Link***.

2.2 Bus_Link Connections

The ***Bus_Link*** port is used for communication with PLC's, SCADA systems and larger telemetry schemes. It is configured through the DCD terminal, and presented in either RS232C or RS485 format.

The RS232 interface is provided through an 8-pin RJ45 socket.

The RS485 interface is provided through a 3-pin connector on ***Micro_Link***, with pins designated A, T and B. The line can be multi-dropped to 32 devices, and should be terminated at both ends with 100Ω.

Note that RS485 is polarity-conscious, so all A terminals must be joined together, as must all B terminals

2.3 Diagnostics

An LED indicator is included on ***Micro_Link*** to monitor the ***Bus_Link*** port. It flashes red when ***Micro_Link*** is transmitting data and green when it is receiving.

Furthermore, ***Micro_Link*** includes very powerful diagnostics via the DCD port. This allows a PC to be connected to configure and monitors all aspects of its operation. When set to monitor the

Bus_Link communications it display every message sent to/from ***Micro_Link***, both in plain English and in hex code.

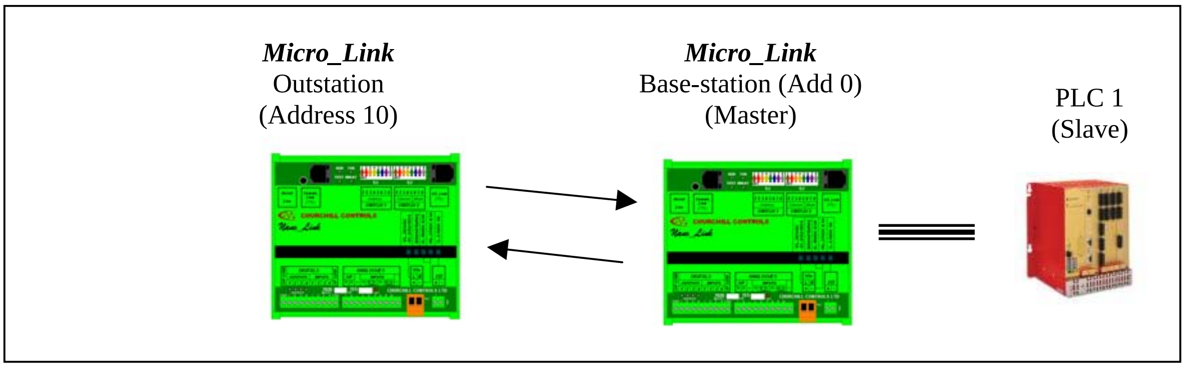
2.4 Use with a PLC Slave

All PLC protocols work on a master-slave principle, whereby one unit on the PLC network is defined as a master and all others are slaves. The master can pass data to/from one or more slaves. Each slave must be given a unique address so the master can distinguish it.

By far the simplest configuration when using ***Micro_Link*** with a PLC is to configure the ***Micro_Link*** as the master and the PLC as a slave. The ***Micro_Link*** will hence be responsible for initiating all transactions.

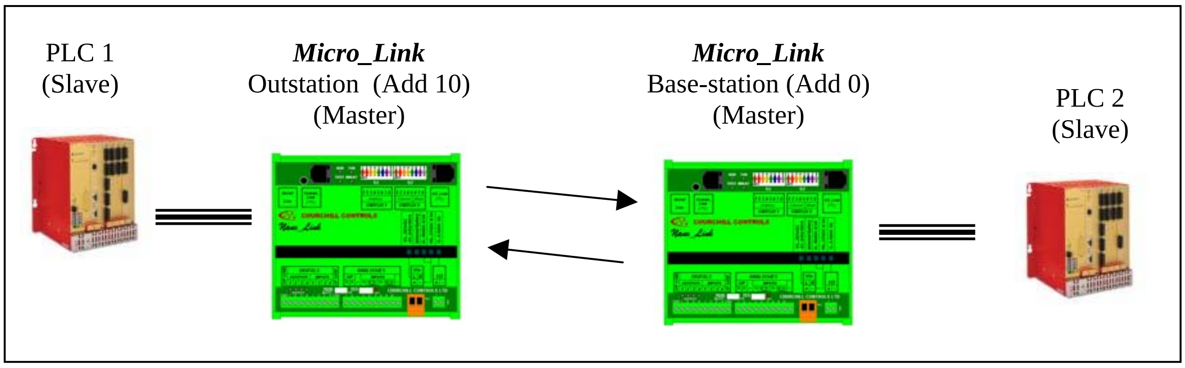
The user need have no knowledge of the internal database structure of the ***Micro_Link***. All data routing is configured in the ***Micro_Link*** Data Routing Table using simple descriptions to define the data source and destination.

Example 1: Transferring real world inputs & outputs to a PLC slave device via a remote base-station.



In this example the base-station's Data Routing Table would be configured to pass I/O from the outstation to/from specific registers in PLC 1. PLC 1 will have a defined address that must be used within the base-station Data Routing Table to identify the PLC. It should be apparent that more than one slave PLC could be connected to the ***Micro_Link*** base-station, provided the electrical interface supports multi-drop usage.

Example 2: Two PLC slaves transferring register data.



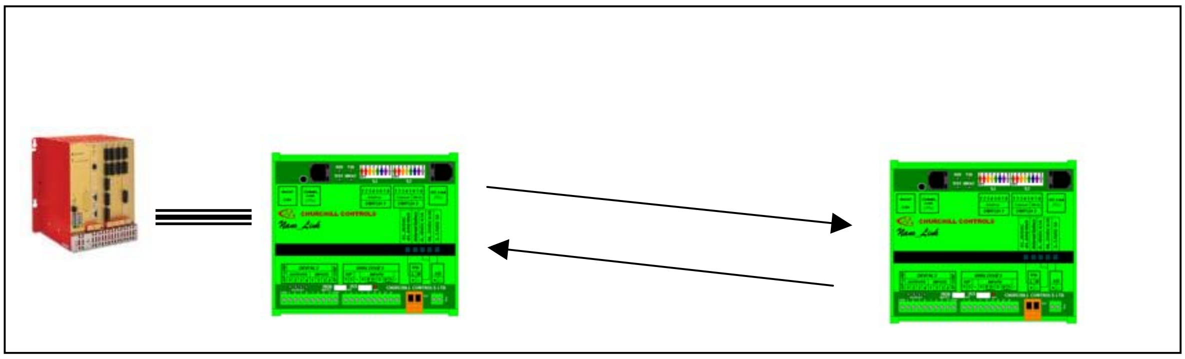
In this example the outstation Data Routing Table would be configured to copy data from PLC 1 to the outstation database, where it will appear as additional I/O. The base-station Data Routing Table can then be configured to pass the outstation's I/O to/from PLC 2.

2.5 Use with a PLC Master

If the PLC cannot be set as a Slave device for any reason, it is possible to set Micro_Link as the Slave, allowing the PLC to be the Master.

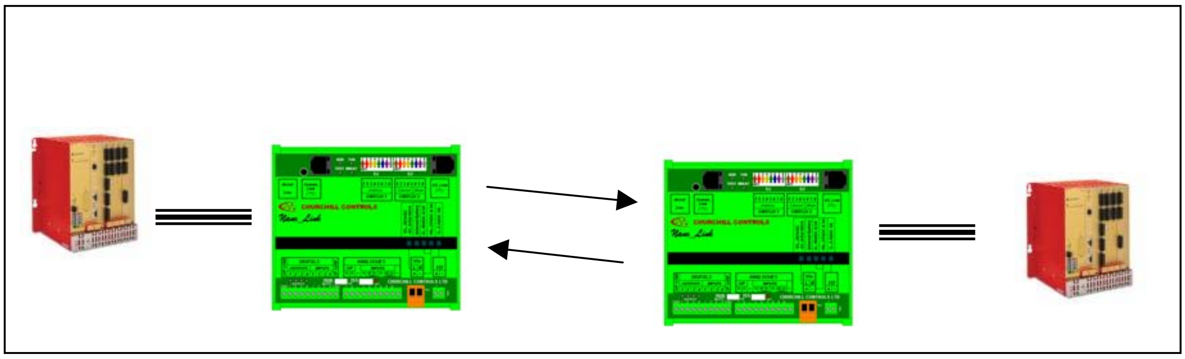
This is more complicated than previously, as the user needs have no knowledge of the internal database structure of the Data_Link system.

Example 3: A PLC master transferring real world inputs & outputs to a remote outstation.



The outstation is configured with two independent addresses. One defines its address on the ***Micro_Link*** network (10 in this example), and the other defines its address on the PLC network. The PLC master must be configured to pass data to/from specific database locations within the outstation. The user needs to understand the structure of the ***Micro_Link*** database to determine which database locations to use.

Example 4: Two PLC masters transferring register data.



In this example PLC 1 will be configured to copy registers to/from its own database into the outstation database. Similarly PLC 2 will be configured to copy the same registers from/to its own database. The ***Micro_Link*** protocol will automatically ensure that the relevant registers within the outstation and the base-station are kept in step.

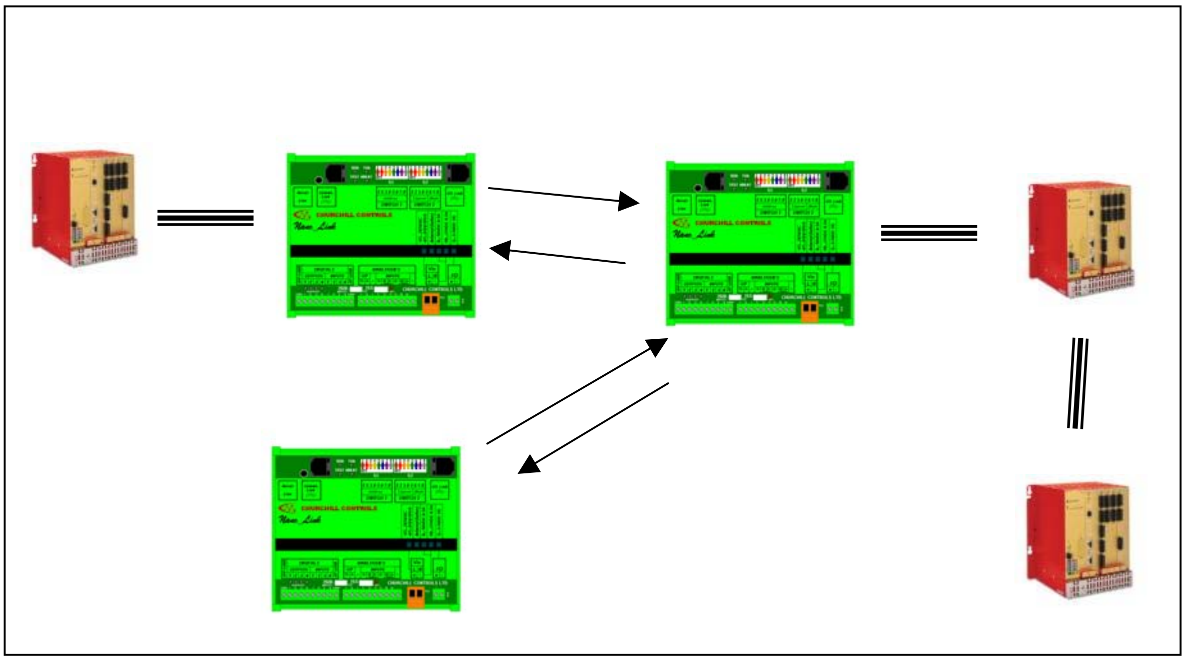
2.6 Use with a Combinations of the Above.

Referring to Example 2 above, it can be seen that there are three independent asynchronous communications subsystems:

- Serial Comms from PLC1 to ***Micro_Link*** base-station
- Radio Comms from ***Micro_Link*** base-station to ***Micro_Link*** outstation
- Serial Comms from PLC2 to ***Micro_Link*** outstation.

As these are each self-contained, it is possible to add to each of them as required, adding further PLCs and outstations as necessary.

Note that since ***Micro_Link*** operates on a polling principle, there can only be one base-station on any given system, with one or more outstations. Similarly, since most PLC protocols also work on a polling principle, there can only be one PLC master on any given system, with one or more slaves:



It should be obvious that in this example the base-station and PLC 3 must be configured with different slave addresses.

2.7 Other Considerations

As well as maintaining within its database all the relevant I/O conditions, each ***Micro_Link*** also monitors it own status. Certain database locations therefore store alarm flags (e.g. comms fail and battery low) and monitoring levels (e.g. battery voltage and radio receive signal strength). The user can include entries in the Data Routing Table to copy these to discrete outputs, but they can equally be copied to any required location within PLCs via the ***Bus_Link*** interface.

3 Hardware Connection

The ***Bus_Link*** port is used for communication with PLC's, SCADA systems and larger telemetry schemes. It is configured through the DCD terminal, and presented in either RS232C or RS485 format.

3.1 RS232C

The RS232 interface is provided through an 8-pin RJ45 socket, configured as follows:

	Bus_Link (RS232) RJ45	**PC 25-way D**	**PC 9-way D**	**Modicon PLC RJ45**
+5V				← 8
NC	1	8-19, 21-25	9	
DSR	2 →	→ 6	→ 6	→ 7
TXD	3 ←	← 2	← 3	← 6
RXD	4 →	→ 3	→ 2	→ 5
0V	5 <>	<> 7	<> 5	<> 4
RTS	6 ←	← 4	← 7	← 3
CTS	7 →	→ 5	→ 8	→ 2
Shield	8 <>	<> 1	<> 1	<> 1
DTR		← 20	← 4	

Columns 3 and 4 show typical connections to a personal computer, while column 5 shows that connection to a Modicon Micro PLC requires only an 8-way RJ45 reverse cable (linking pin 1 - 8, 2 - 7, 3 - 6, etc.).

The most common problem when connecting to a serial device to ***Micro_Link*** is the orientation of the TXD and RXD lines. ***Micro_Link*** terminology regards TXD as the line through which the remote device sends data, but some manufacturers regard it as the line on which ***Micro_Link*** would send data. The simplest way to identify the orientation is to measure the voltage on the relevant pin (relative to the 0V pin) when the units are not connected together. Each output line will measure a voltage in excess of 5V (it may be positive or negative), while each input pin will not measure any voltage. Obviously the output from one device must be connected to the input of the other.

Another problem that may occur relates to handshaking. ***Micro_Link*** does not need any handshaking, since it is able to receive data at all times. Most third party products are the same, but some will raise RTS when they need to send data, and delay sending it until CTS is raised. This can be achieved by connecting together the RTS and CTS pins on the PLC. Furthermore, some devices raise DTR to signify that they are active, and will ignore all communication until DSR is returned. This can be achieved by connecting together the DTR and DSR pins on the PLC.

Note that RS232 is a point-point interface, so can only be used to connect ***Micro_Link*** to a single PLC device.

3.2 RS485

RS485 is a two-wire interface that allows up to 32 devices to be interconnected via a single twisted pair of wires. Each end of the wire should be terminated with a 100Ω resistor. A termination resistor is fitted within ***Micro_Link***, but should only be connected if ***Micro_Link*** is at one end of the wire. A 3-pin connector is therefore provided on **Micro_Link**, with pins designated A, T and B. The line should be connected between A and B, and the termination resistor can be activated by linking T to B.

Note that RS485 is polarity-conscious, so all 'A' terminals must be joined together, as must all 'B' terminals. Some manufacturers may use a different designation, but a clear indication is provided on ***Micro_Link*** to show the correct orientation:

The most common problem when connecting to a serial device is the orientation of the TXD and RXD lines, as this notation can vary between instrument manufacturers.

If the A and B connections are reversed, the ***Bus_Link*** LED on ***Micro_Link*** will show steady green. If this happens, simply reverse the connections.

3.3 Hardware Diagnostics

Before attempting to design the configuration in detail, the user is advised to install a very simple configuration first, to confirm that the communication path is working. A single line, copying a register from the ***Micro_Link*** to the PLC, is adequate.

If the communications is working the ***Bus_Link*** LED should be normally off, but flash red whenever ***Micro_Link*** sends data, and green when it receives data. If the PLC includes a comms monitor LED it should also flash as data is passed.

If the LED's suggest no communication is taking place, the user should address the problem before proceeding. The most likely problem is the wiring, as described above. Once the wiring is correct, the next hurdle is to confirm that the ***Micro_Link*** and the PLC understand each other, so the data rate and format must match, and the correct slave address must be used.

Once the user has convinced himself that the wiring and the protocol are correct, he can run DCD diagnostics (as described later in this document) to watch the data.

4 Software Configuration

4.1 General

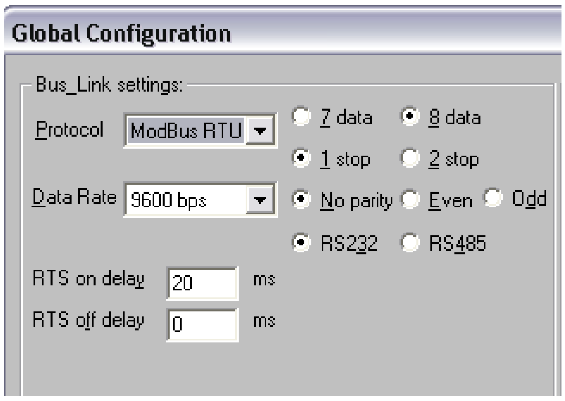
All software configuration of ***Bus_Link*** is carried out using the ***Data_Link Configuration and Diagnostic*** (DCD) software that is supplied on a CD with every ***Micro_Link*** system. Alternatively it can be downloaded from our website on www.churchill-controls.co.uk .

When DCD is running, a configuration can be created either by opening an existing file, creating a new file or uploading the configuration from a ***Micro_Link***. Each configuration creates a new window.

4.2 Protocol

When linking the two pieces of equipment together, their data formats must be matched. That is, both items must be working on the same protocol, at the same data speed, using the same number of data bits, the same number of stop bits and the same parity.

These settings can be chosen by clicking on the 'Global' button at the bottom of the configuration window. This will open a dialog box, the top left part of which is as follows:



Global Configuration

Bus_Link settings:

Protocol: ModBus RTU

Data Rate: 9600 bps

7 data [✓ 8 data]

[✓ 1 stop] 2 stop

[✓ No parity] Even Odd

[✓ RS232] RS485

RTS on delay: 20 ms

RTS off delay: 0 ms

The parameters shown should be self-explanatory, and must match those of the PLC.

4.3 Micro_Link Database

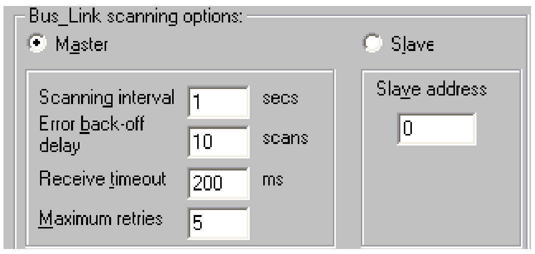
Each ***Micro_Link*** unit maintains a database of 2000 input registers, 2000 output registers, 8000 digital inputs and 8000 digital outputs. ***Micro_Link*** maps all the I/O from the base-station and outstations into this database is a well-defined, structured way. DCD knows the structure and simplifies identification of each I/O point into simple plain English terminology, such as 'Outstation 10 Digital Input 1'.

If ***Micro_Link*** is configured as a ***Bus_Link*** master, the user does not need any knowledge of the mapping, since he will be configuring the ***Bus_Link*** communications through DCD. However, if ***Micro_Link*** is configured as a ***Bus_Link*** slave, the user does need to know the database format so he can access the relevant registers.

It is therefore much easier to configure systems that use ***Micro_Link*** as a master

4.4 Use with a PLC Slave

The ***Micro_Link*** mode is configured using the lower left part of the dialog box:



Bus_Link scanning options:

[✓ Master] Slave

Scanning interval: 1 secs

Error back-off delay: 10 scans

Receive timeout: 200 ms

Maximum retries: 5

Slave address: 0

To configure a Micro_Link as a Master, click the Master button as shown. The parameters in the box below the button are only relevant when ***Micro_Link*** is configured as a master:

Scanning Interval: The period at which the master polls the slave(s) for information

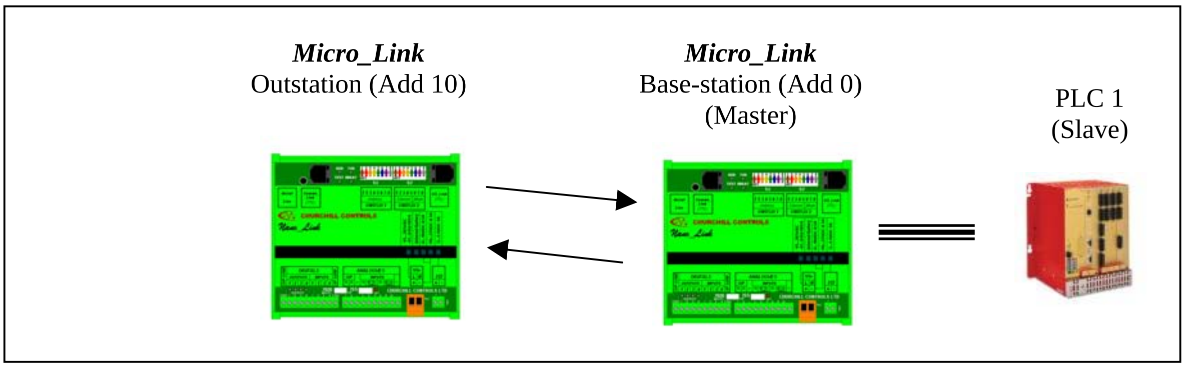
Error back-off delay: In the event of no response from a slave, this is the number of scans that are subsequently skipped until the master tries to poll the slave again.

Receive Timeout: This is the time for which the master will wait for a reply from the slave

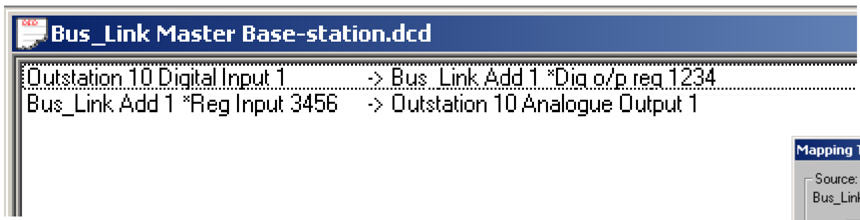
Maximum retries: The number of times the master will retry the same command to the slave before going into the back-off delay.

4.5 Data Routing when using a PLC Slave:

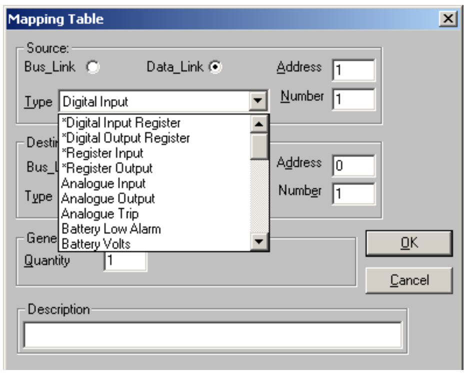
Example 1: Transferring real world inputs & outputs to a PLC slave device via a remote base-station.



As stated above, when ***Micro_Link*** is configured as a master device, all data routing is configured via DCD. A sample Data Routing table is shown for the base-station:

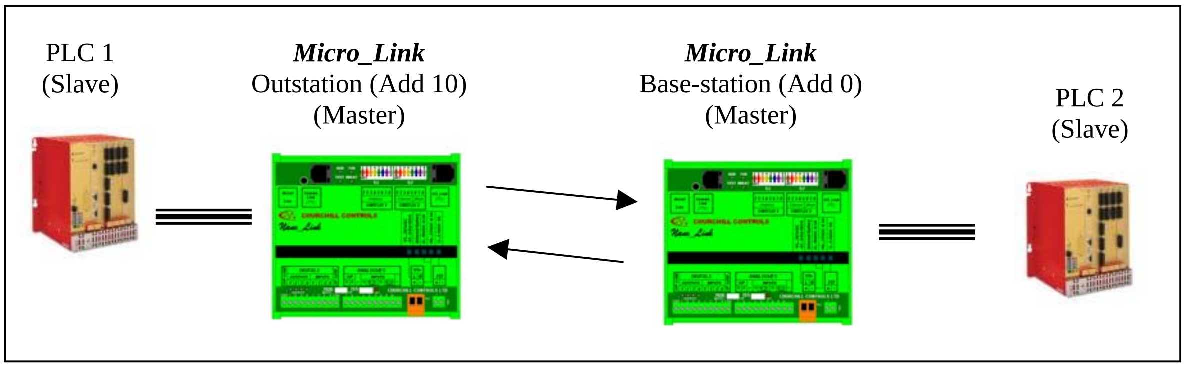

Bus_Link Master Base-station.dcd

```
Outstation 10 Digital Input 1        -> Bus_Link Add 1 *Dig o/p reg 1234
Bus_Link Add 1 *Reg Input 3456       -> Outstation 10 Analogue Output 1
```



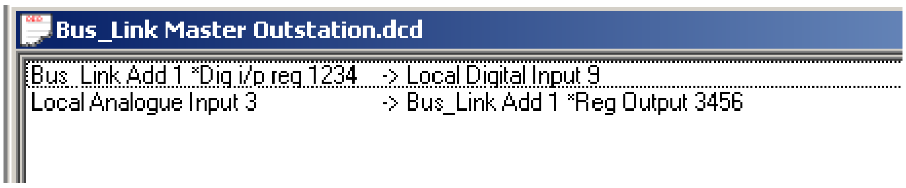
This will copy the state of digital input 1 at the outstation to digital output register 1234 in the PLC, and copy output register 3456 from the PLC to the first analogue output at outstation 10. Note that references to registers in the PLC use absolute addressing (i.e. the definitions at the top of the pull-down menu, prefixed with *), since PLC's are not aware of the data structure used in ***Micro_Link***:

Example 2: Two PLC slaves transferring register data.



This requires entries in the Data Routing Table at both the outstation and the base-station:

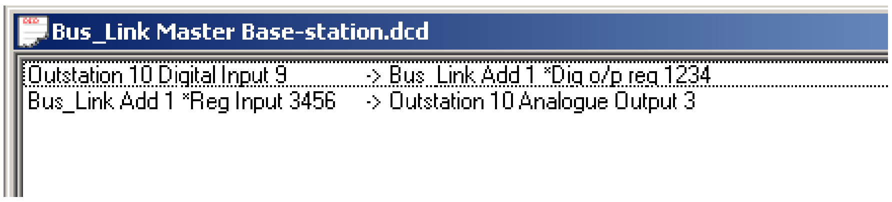
Outstation:


Bus_Link Master Outstation.dcd

```
Bus_Link Add 1 *Dig i/p reg 1234    -> Local Digital Input 9
Local Analogue Input 3              -> Bus_Link Add 1 *Reg Output 3456
```

Because data is copied from the PLC to digital input 9, this outstation will appear to the base-station as if it has a total of 9 digital inputs (i.e. the 8 inputs that are integral to ***Micro_Link***, plus one derived from the PLC). It is not relevant to the base-station whether the extra inputs come from expansion modules at the outstation or from ***Bus_Link***. Similarly, the outstation will appear to have an extra analogue output in addition to the two which are integral to ***Micro_Link***.

Base-station:



```
Outstation 10 Digital Input 9       -> Bus_Link Add 1 *Dig o/p reg 1234
Bus_Link Add 1 *Reg Input 3456      -> Outstation 10 Analogue Output 3
```

The corresponding configuration at the base-station could be:

The overall effect of this will be to copy digital input 1234 of the PLC at the outstation to digital output register 1234 of the PLC at the base-station, and to copy input register 3456 of the PLC at the base-station to output register 3456 of the PLC at the outstation.

4.6 Data Routing when using a PLC Master:

If the PLC cannot be set as a slave device, ***Micro_Link*** can be configured as the slave, allowing the PLC to be the master. This is more complicated than previously, as the user needs have no knowledge of the internal database structure of the ***Micro_Link***.

A ***Micro_Link*** base-station effectively maintains a database of 2000 input registers, 2000 output registers, 8000 digital inputs and 8000 digital outputs. This can be more conveniently viewed as 250 input data blocks and 250 output datablocks.

A ***Nano_Link*** outstation only occupies the root data block, even if it is equipped with its full compliment of expansion modules. The function of the digitals and registers used by it are as follows:

Nano_Link Root Data Block Usage

	INPUT DATA BLOCK Digital	Register		OUTPUT DATA BLOCK Digital	Register
0	Comms Fail alarm	Totalised count for digital input 1	0	-	-
1	Battery Low alarm			-	
2	Spare (read as 1)			-	
3	Spare (read as 1)			-	
4	Spare (read as 1)	Totalised count for digital input 2	1	-	-
5	Mains Fail			-	
6	Spare (read as 1)			-	
7	Spare (read as 1)			-	
8	Digital Input 1	Totalised count for digital input 3	2	Digital Output 1[1]	-
9	Digital Input 2			Digital Output 2[1]	
10	Digital Input 3			Digital Output 3[1]	
11	Digital Input 4			Digital Output 4[1]	
12	Comms Fail alarm	Totalised count for digital input 4	3	-	-
13	Battery Low alarm			-	
14	Mains Fail			-	
15	Spare (read as 1)			-	
16	Digital Input 9[2]	Battery Volts	4	Digital Output 9[3]	-
17	Digital Input 10[2]			Digital Output 10[3]	
18	Digital Input 11[2]			Digital Output 11[3]	
19	Digital Input 12[2]			Digital Output 12[3]	
20	Digital Input 13[2]	Radio Receiver Signal Strength (RSSI)	5	Digital Output 13[3]	-
21	Digital Input 14[2]			Digital Output 14[3]	
22	Digital Input 15[2]			Digital Output 15[3]	
23	Digital Input 16[2]			Digital Output 16[3]	
24	Digital Input 17[2]	Analogue Input 1	6	Digital Output 17[3]	Analogue Output 1[1]
25	Digital Input 18[2]			Digital Output 18[3]	
26	Digital Input 19[2]			Digital Output 19[3]	
27	Digital Input 20[2]			Digital Output 20[3]	
28	Digital Input 21[2]	Analogue Input 2	7	Digital Output 21[3]	Analogue Output 2[1]
29	Digital Input 22[2]			Digital Output 22[3]	
30	Digital Input 23[2]			Digital Output 23[3]	
31	Digital Input 24[2]			Digital Output 24[3]	

All alarm bits are ‘1’ in the normal state and ‘0’ when in alarm

NOTES

[1] Only available if the ***Nano_Link*** is fully equipped

[2] Only available if a digital input expansion module is fitted, otherwise read ‘0’

[3] Only available if a digital output expansion module is fitted, otherwise ignored

A ***Micro_Link*** outstation or base-station equipped with up to one digital input expansion module and/or one digital output expansion module also occupies only the root data block:

Micro_Link Root Data Block Usage

	INPUT DATA BLOCK			**OUTPUT DATA BLOCK**	
	Digital	**Register**		**Digital**	**Register**
0	Comms Fail alarm	Totalised count for digital input 1	0	-	-
1	Battery Low alarm			-	
2	Hardware Fail alarm			-	
3	**Bus_Link** fail alarm			-	
4	Complete Comms Fail	Totalised count for digital input 2	1	-	-
5	Mains Fail			-	
6	Spare (read as 1)			-	
7	Spare (read as 1)			-	
8	Digital Input 1	Totalised count for digital input 3	2	Digital Output 1	-
9	Digital Input 2			Digital Output 2	
10	Digital Input 3			Digital Output 3	
11	Digital Input 4			Digital Output 4	
12	Digital Input 5	Totalised count for digital input 4	3	Digital Output 5	-
13	Digital Input 6			Digital Output 6	
14	Digital Input 7			Digital Output 7	
15	Digital Input 8			Digital Output 8	
16	Digital Input 9[1]	Battery Volts	4	Digital Output 9[2]	-
17	Digital Input 10[1]			Digital Output 10[2]	
18	Digital Input 11[1]			Digital Output 11[2]	
19	Digital Input 12[1]			Digital Output 12[2]	
20	Digital Input 13[1]	Radio Receiver Signal Strength (RSSI)	5	Digital Output 13[2]	-
21	Digital Input 14[1]			Digital Output 14[2]	
22	Digital Input 15[1]			Digital Output 15[2]	
23	Digital Input 16[1]			Digital Output 16[2]	
24	Digital Input 17[1]	Analogue Input 1	6	Digital Output 17[2]	Analogue Output 1
25	Digital Input 18[1]			Digital Output 18[2]	
26	Digital Input 19[1]			Digital Output 19[2]	
27	Digital Input 20[1]			Digital Output 20[2]	
28	Digital Input 21[1]	Analogue Input 2	7	Digital Output 21[2]	Analogue Output 2
29	Digital Input 22[1]			Digital Output 22[2]	
30	Digital Input 23[1]			Digital Output 23[2]	
31	Digital Input 24[1]			Digital Output 24[2]	

All alarm bits are '1' in the normal state and '0' when in alarm

<u>NOTES</u>

[1] Only available if a digital input expansion module is fitted, otherwise read '0'

[2] Only available if a digital output expansion module is fitted, otherwise ignored

If more than one digital input expansion module is fitted it will 'spill over' into the next input data block, as will any analogue input expansion modules. Similarly, if more than one digital output expansion module is fitted it will 'spill over' into the next output data block, as will any analogue output expansion modules. A ***Micro_Link*** base-station or outstation could therefore occupy up to 33 consecutive data blocks in the worst case (i.e. if it is fitted with a full compliment of 32 8-channel analogue input modules).

Any given register or digital can be identified by an absolute address or a relative address from a reference Data Block.

For example, each of the following refers to the same register:

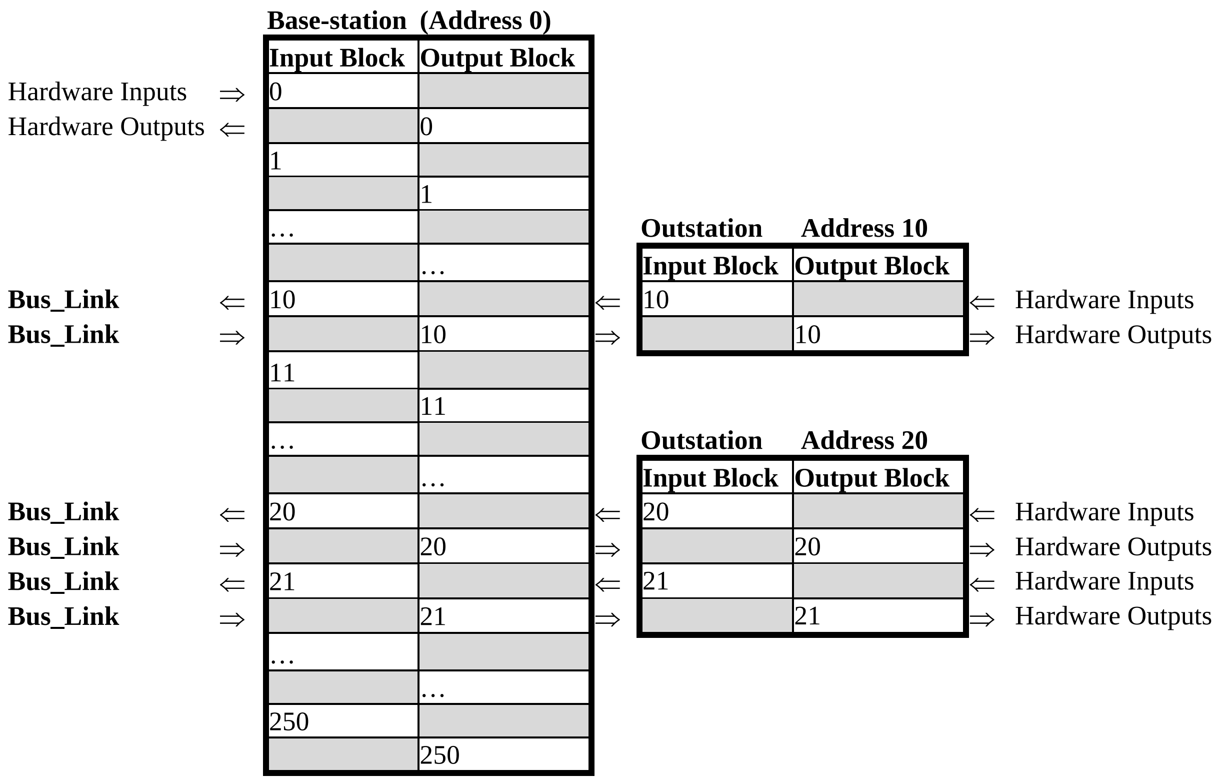
a) Input Register 172
b) Input Block 0, Register 172
c) Outstation Address 21, Battery Volts (21 x 8 + 4)
d) Outstation Address 20, Analogue Input 6 (20 x 8 + 12)

Absolute addressing (format (a)) is the format used by PLC's and SCADA systems communicating via serial protocols such as Modbus.

All the other formats identify the register relative to a given start point. Relative addressing is convenient for configuring ***Micro_Link***, since the Root Data Block can be used to identify the relevant outstation address, with the register number identifying the relevant digital or register within the outstation.

Note that the ***Micro_Link*** numbering convention always starts from 0. Some PLC protocols (e.g. Modbus) start counting from 1, so the register in the above example would be regarded by some Modbus systems as register 173.

A ***Micro_Link*** base-station effectively maintains a complete database of 2000 input registers, 2000 output registers, 8000 digital inputs and 8000 digital outputs. This can be more conveniently viewed as 250 input data blocks and 250 output datablocks. ***Micro_Link*** protocol automatically transfers the relevant parts of the base-station database to/from the corresponding areas within the database at each outstation:

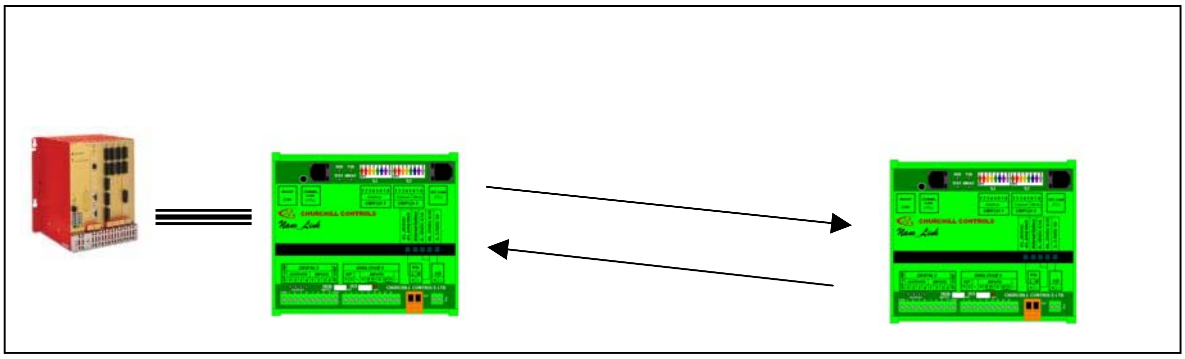


The PLC master will need be configured to identify which data it requires to access, using absolute addressing.

For example, to read outstation 20 battery volts the SCADA system should read Output Register 164 at the base-station (20 x 8 + 4).

Any register or digital can be copied to any destination(s) via **Bus_Link** and/or internal transfers. However, any given output register or digital must only be fed data from one source.

Example 3: A PLC master transferring real world inputs & outputs to a remote outstation.

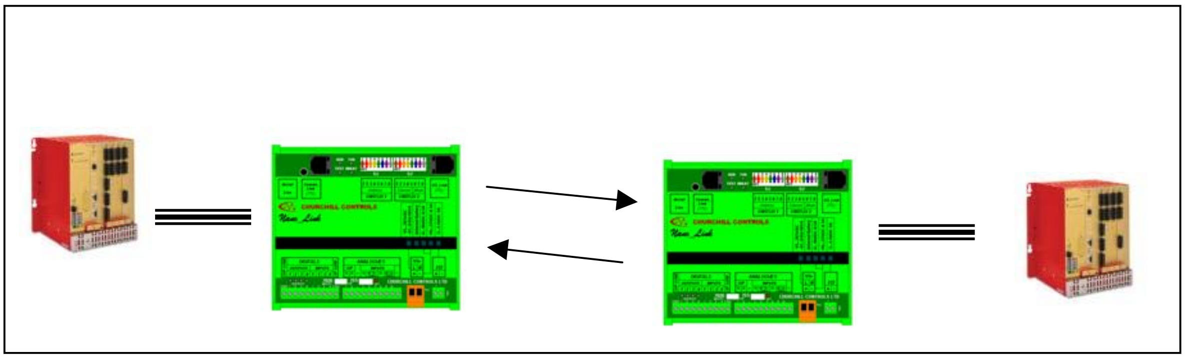


No data routing table is required for the base-station since the PLC Master will initiate all communications, by interrogating the desired registers within the Data_Link database. Once the base-station sees a request from the PLC, if it is required to get the required information from a remote outstation it will automatically begin polling that outstation. It will continue polling the outstation until it is powered down or its configuration is changed, regardless of whether or not the PLC continues to poll it.

For example, to read Digital Inputs 1…8 at the outstation, the PLC must read digital inputs 328…335 from the base-station. (Since there are 32 digitals per data block, outstation 10 uses digitals 320…351. Inputs 320…327 are alarm flags, so the outstation inputs start at digital 328.)

Similarly, to write to Outstation 10 Analogue Output 1, the PLC must write to output register 86. (Since there are 8 registers per data block, outstation 10 uses registers 80…87. The first 4 registers are totalised counts on the first 4 digital inputs, followed by the outstation battery volts, then the outstation RSSI, then the 2 internal analogue inputs.)

Example 4: Two PLC masters transferring register data.



Data routing tables are not needed in either of the ***Micro_Link***'s. Each PLC must copy data to/from registers within the ***Micro_Link*** that are not used for hardware I/O. In this example only data blocks 0 and 10 are used by hardware, so the PLC's could theoretically map data to any registers outside of these. However, the normal convention is to map data to the registers immediately following those used by the outstation. This ensures that there is minimal risk of duplication of register allocations if the system is subsequently expanded.

5 Comms Fail and Other Alarms

It is usually desirable to have some way of determining whether the Radio (or Line) link between the base-station and outstation is healthy.

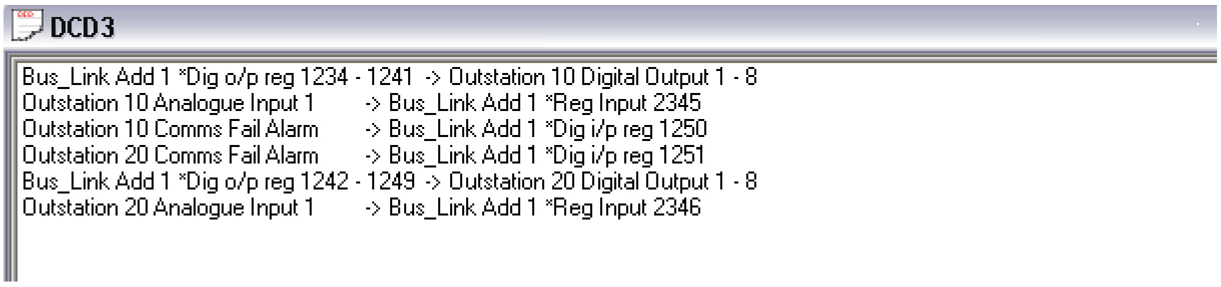
The standard way of achieving this is to map the 'Comm Fail' flags from each outstation to local digital outputs on the base-station. Alternatively, the same flags may be accessed via Bus_Link.

Referring to the data block definitions given above, the alarm flags for each outstation occupy the first 8 digital input registers. For example, to read Comms Fail for Outstation 10, the PLC must read digital input register 320 (Since there are 32 digital per data block, outstation10 starts at 320. Comms Fail ins the first digital in the data block.)

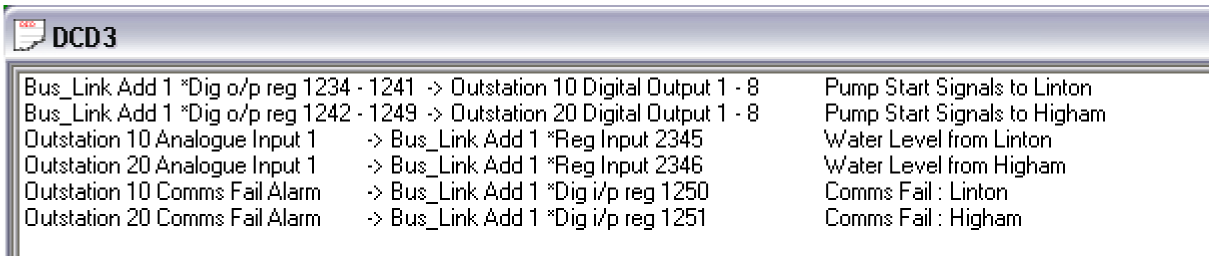
6 Data Routing Table Conventions

When compiling a long data routing table consisting of multiple register transfers, it is usually desirable to ensure that the right hand column follows a defined sequence. This make it immediately apparent if two lines write conflicting data to a given output. It is also good practice to use the description field to annotate each table entry, again for ease of debugging later on.

Below are shown two Data Routing tables. These will both function identically, however the second table is easier to read and manage.



```
Bus_Link Add 1 *Dig o/p reg 1234 - 1241  -> Outstation 10 Digital Output 1 - 8
Outstation 10 Analogue Input 1        -> Bus_Link Add 1 *Reg Input 2345
Outstation 10 Comms Fail Alarm        -> Bus_Link Add 1 *Dig i/p reg 1250
Outstation 20 Comms Fail Alarm        -> Bus_Link Add 1 *Dig i/p reg 1251
Bus_Link Add 1 *Dig o/p reg 1242 - 1249  -> Outstation 20 Digital Output 1 - 8
Outstation 20 Analogue Input 1        -> Bus_Link Add 1 *Reg Input 2346
```



```
Bus_Link Add 1 *Dig o/p reg 1234 - 1241  -> Outstation 10 Digital Output 1 - 8      Pump Start Signals to Linton
Bus_Link Add 1 *Dig o/p reg 1242 - 1249  -> Outstation 20 Digital Output 1 - 8      Pump Start Signals to Higham
Outstation 10 Analogue Input 1        -> Bus_Link Add 1 *Reg Input 2345             Water Level from Linton
Outstation 20 Analogue Input 1        -> Bus_Link Add 1 *Reg Input 2346             Water Level from Higham
Outstation 10 Comms Fail Alarm        -> Bus_Link Add 1 *Dig i/p reg 1250           Comms Fail : Linton
Outstation 20 Comms Fail Alarm        -> Bus_Link Add 1 *Dig i/p reg 1251           Comms Fail : Higham
```

7 Diagnostics

If a PC running DCD is plugged into the DCD port on the ***Micro_Link***, the user can start diagnostics by clicking on the button that has a picture of a computer to open a terminal emulator. Pressing ‘B’ followed by Return will start ***Bus_Link*** diagnostics. If all is well the computer should display something like:

```
Forcing coil 1234 of Modbus address 1
00:00:08 cmd: 01 0F 04 D2 00 01 01 01 56 C1
00:00:08 rep: 01 0F 04 D2 00 01 35 02 O.K.
Reading holding register 3456 of Modbus address 1
00:00:08 cmd: 01 03 0D 80 00 01 87 4E
00:00:08 rep: 01 03 02 00 00 B8 44 O.K.
```

This would be the communications resulting from the configuration given in Example 1. If ***Micro_Link*** is configured as a slave device, such as Example 3, the display would be:

```
00:00:25 cmd: 01 01 01 48 00 08 BC 26 Read coil status 328..335
00:00:25 rep: 01 01 01 00 51 88
00:00:25 cmd: 01 10 00 56 00 01 02 00 02 2B A7 Preset multiple registers 86
00:00:25 rep: 01 10 00 56 00 01 E1 D9
```