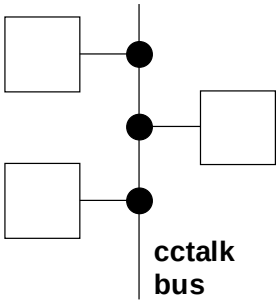


cctalk Serial Communication Protocol
-
Condor Plus Specification
-
Issue 3.2

Company Address

Money Controls Ltd (referred to throughout as MCL)

Please pass any comments you have on the cctalk specification to...

Andy Barson,
Software Manager,
Money Controls Ltd,
New Coin Street,
Royton,
Oldham OL2 6JZ.
ENGLAND.

Tel: 44 (0) 161 678 0111
Fax: 44 (0) 161 628 5860

E-mail: AndyB@moneycontrols.com

Web Site: http://www.moneycontrols.com/

Please note that all product related enquiries should be referred to the Money Controls Technical Services Department.

Revision History

Issue	Date	Comments
1.0	29-04-96	Draft specification
2.0 Beta 1	31-07-96	Major revision and expansion (new section numbers)
2.0 Beta 2	15-08-96	Addition of Appendix 23 & Figure 4
2.0 Beta 3	28-08-96	Addition of Figure 5
2.0 Beta 4	02-09-96	Addition of another error code (see Figure 5)
		Section 12 : Well-written slaves devices...
2.0 Beta 5	20-09-96	Header 217 'Request tube status' changed to
		'Request payout high / low status' for clarity
		Header 208 'Request hopper status' changed to
		'Request payout absolute count' for clarity
		Addition of Header 207 'Set payout absolute count'
		Addition of Header 206 'Empty payout'
		Addition of Header 205 'Request audit information block'
		Acknowledgement of 'cctalk' as trademark
		Web site address
		Update of all hopper / tube / payout commands
		Table / Figure split
		More category strings defined (Table 2)
		More fault codes defined (Table 4)
		Add section 24 - Naming Convention
		Update of section 6, connector details
		Mention of DIL switches for address setting (section 16.1)
2.0	18-06-97	Table 1 updated to issue 4
		Table 2 updated to issue 3
		Table 4 updated to issue 2
		Logical support for up to 254 slave devices
		Mention of PIC16C55 implementation
		Possible operation at 9600 baud
		Removal of connector type 2a
		Removal of 'reserved destination addresses' complication
		Definition of NAK message (reserved header 5)
		Revision to section 16.1
		Addition of Table 5 - cctalk Status Codes
		Addition of Table 6 - cctalk Commands by Category
		16.2.2 - general update and extra flexibility in return data
		Reduced form of 'Read buffered credit or error codes'
		Revision to A5 - Naming Convention
		All references to 'window' changed to 'coin position'
3.0 Beta 1	20-05-98	Minor descriptive text changes
		Connector types changed to match current product range
		Table 1 updated to issue 5
		Table 2 updated to issue 4 - inclusion of typical default addresses
		Table 3 updated to issue 3
		Table 4 updated to issue 3
		Table 5 updated to issue 2
		Table 6 updated to issue 2
		Appendix A5 (Naming Convention) extended
		Update to 16.1 (Implementation Levels)
		Definition of BUSY message (header 6)
		Addition of ‘Teach mode control’
		Addition of ‘Teach mode status’
		Addition of ‘Calculate ROM checksum’
		Addition of ‘Request build code’
		Addition of ‘Keypad control’
		Addition of ‘Request payout status’
		Addition of ‘Modify default sorter path’

		Addition of 'Request default sorter path'
		Addition of 'Modify payout capacity'
		Addition of 'Request payout capacity'
		Addition of 'Modify coin id'
		Addition of 'Request coin id'
		Addition of 'Upload window data'
		Addition of 'Download calibration info'
		Addition of 'Modify coin security setting'
		Addition of 'Request coin security setting'
		Addition of 'Modify bank select' etc.
		Modification to 'Dispense coins'
		Modification to 'Dispense change'
		Modification to 'Empty payout'
		Modification to 'Modify payout absolute count'
		Modification to 'Request payout absolute count'
		Modification to 'Request audit information block'
		'Request build version' changed to 'Request database version'
		More details on 'Upload coin data'
		Rename 'Request electronic serial number in binary'
		as 'Request serial number'
		Addition of Section 9.2 - The BUSY Message
		Addition of Section 8.6.1 - Checksum Theory
		Addition of Section 5.1 - RS485 Drivers
		Addition of Section 13.1 - Retransmission
		Addition of Section 17.1 - Polling Algorithms
		Addition of A5.1 - MCL Product Examples
		Addition of Table 7 - cctalk Coin Upload Reply Codes
		Support for inhibited coin recognition in error code table
		'Keypad control' - addition of cursor position byte
		'Display control' - addition of display size query
3.0 Beta 2	11-11-98	Reference to BACTA industry-standard serial specification
		Clarification of 'implementation level'
		Extra hopper commands
		Addition of 'Request thermistor reading'
		Modification to standard category string table
3.0	21-12-98	Released
	17-03-99	Reference to 'Upload flash memory' removed
3.1	15-04-03	Added TSP number
		Amended table 1 to include text in 'supported' column
		Modified headers and footers
3.2	30-06-04	Modified footers

Contents

Appendices

Figures

Tables

1 Introduction

1.1 Serial versus Parallel : A Coin Industry Perspective

Both serial and parallel interface techniques have advantages and disadvantages. Parallel interfaces are fast and in some applications provide the simplest way of transferring information. However, cable harnessing costs can reach a significant proportion of the original equipment costs as the number of data lines increase. Problems with crimp connectors and dry solder joints can give reliability problems when a large number of wires are used to send data. Serial interfaces on the other hand reduce cabling costs to the minimum and often enable extra features (such as self-testing and expansion) to be incorporated into a product with very little overhead. Serial interfaces also provide a simple and efficient way of connecting two or more devices together in situations which would be totally impractical with a parallel interface. This reduces cabling costs further in applications which require a number of devices to be connected to a single host controller.

The cash handling industry now embraces many different types of technology from coin and token acceptors through bill validators and magnetic / smart card readers to intelligent payouts and changers. A way of connecting all these different types of peripherals in a simple and consistent manner is a stated aim of many manufacturers and a serial bus is the obvious solution.

1.2 What is cctalk ?

cctalk (lower-case, pronounced see-see-talk) is the MCL serial communication protocol for low speed control networks. It was designed to allow the interconnection of various types of cash handling and coin validation equipment on a simple 2-wire interface (data and ground). A simple application consists of one host controller and one peripheral device. A more complicated application consists of one host controller and several peripheral devices with different addresses. Although multi-drop in nature, it can be used to connect just one host controller to one slave device.

The protocol is really concerned with the high level formatting of bytes in an RS232-type data stream which means that it is immediately accessible to a huge range of applications throughout the control industry. There is no requirement for custom integrated circuits, ASIC's, special cables etc. It is cheap to manufacture and easy to implement.

The protocol was created from the *bottom end up*. Rather than starting with a full-blown networking or vending protocol and cutting out the features which weren't needed, it was developed from a simpler RS232 format in use at MCL for many years. This means it is a protocol ideal for use in the coin industry with no *excess fat*. There are no complicated logging-on or transaction processing sequences to go through. It does a simple job with the minimum of fuss. Although developed within the coin industry, it has obvious potential in many engineering fields and is flexible enough to be expanded indefinitely.

A significant advantage of using RS232 as the base format is that the protocol can easily take place between remote sites on existing telephone lines with the addition of a modem at each end. With the introduction of single-chip modem technology, more and more applications are benefiting from remote programming capability. The cctalk language is the same no matter what the distance between host controller and slave device. Some loop delays may be longer but that can easily be allowed for when writing the software. The most talked about areas in coin handling are the remote programming of new coin sets and the remote FLASH upgrading of firmware.

1.3 What are its Capabilities and Features ?

cctalk has a byte-oriented message structure rather than a bit-field message structure which means that most logical *limits* of the software are 255 or nearly 255. This provides plenty of scope for most control networks. Although byte structures take up slightly more memory, they are usually much easier to implement on 8-bit micro-controllers.

The logical addressing of cctalk allows up to 254 slave devices to be connected to a single host controller. The addresses of the slave devices do not determine the equipment type - it is perfectly possible to have 3 identical bill validators attached to the network with different logical addresses.

An 8-bit data structure is used throughout the protocol - in RS232 parlance there is no requirement for a 9th *address* or *wake-up* bit. This simplifies a lot of communication software.

Rather than have a few commands which return large packets of data (in excess of 30 bytes), the protocol encompasses a much larger number of smaller and more efficient commands. For instance, if you request a device serial number then that is exactly what you get - you do not have to wade through packets of build numbers, version numbers and null fields until the data you are interested in finally arrives. Our experience at MCL tells us that customers have widely differing requirements and this approach is best - let them pick and choose from an extensive command list.

Variable message lengths are supported. This allows a convenient way of returning ASCII strings back to the host controller. An example of where this is useful is when the host controller seeks the identity of an attached peripheral device. This may be a request for the manufacturer name, equipment category or product code.

Security has been built into the protocol from the outset. There is a mechanism whereby certain commands may be PIN number protected. For instance, there is a command which allows the storage of customer specific data. If this is commercially sensitive data then it can be read / write protected with a 32-bit PIN number.

1.4 Is it Difficult to Implement ?

cctalk is designed to run efficiently on low cost 8-bit micro-controllers with very limited amounts of RAM and ROM. Even so, there is no reason why a successful 4-bit microcontroller version could not be implemented. The software overhead required to support this serial protocol on a product is very small (typically < 2K of code for the core commands).

To support this protocol on an 8-bit microcontroller typically requires :

- 1K to 3K of ROM
- 30 bytes to 200 bytes of RAM
- 1 x UART
- 1 x 16-bit timer

Some kind of non-volatile memory such as EEPROM is useful for the storage of configurable parameters.

MCL has written cctalk modules for the Motorola 68HC05 family and the Hitachi H8 family.

It is possible to implement a cut-down version of the software on a simple PIC processor such as the PIC16C55. This tiny device has...

- 512 bytes of ROM
- 24 bytes of RAM
- Simple 8-bit timer
- No interrupts, no UART

The serial communication software may be interrupt-driven or polled. At 4800 baud, each byte transmitted or received takes 2ms. Therefore, a typical microcontroller application may be written with a poll of the serial port every 1ms. This will guarantee that no receive data is ever missed and is slow enough to ensure the main application is not compromised. However, there are some applications where an interrupt-driven serial port is the best choice. Both approaches are compatible with cctalk.

1.5 Are there any Royalties to Pay or Licences to Obtain ?

No, because cctalk is an **open standard**. The word 'cctalk' has been registered as a European trademark and it may be used to designate conformance to this protocol on product labels and in manuals. No other use of this trademark is acceptable.

Standards are currently controlled by MCL and all original specifications are produced here. Comments and suggestions are welcomed from all interested parties and we will try to release future versions of the standard which meet as many new requirements as possible.

1.6 A Quick Example of a cctalk Message

Here is a typical cctalk exchange for a host controller requesting the serial number of an attached peripheral.

Host sends 5 bytes : [2] [0] [1] [242] [11]
Peripheral returns 8 bytes : [1] [3] [2] [0] [78] [97] [188] [143]

Serial number = 12,345,678

A total exchange of 13 bytes produces the serial number - no other bus traffic is necessary.

2 Notation

The **master** device initiates a communication sequence and sends a command or a request for data. The words *master*, *host* and *server* are interchangeable in this document. Example masters include PC, VMC and MPU.

The **slave** device responds to a command with an acknowledge or a message containing 1 or more data bytes. The words *slave*, *guest* and *client* are interchangeable in this document. Example slaves include Coin Acceptor or Mech.

All data byte values below are shown in decimal unless stated otherwise.

Values between square brackets, [XXXXXXXX], indicate message bytes, i.e. 8 bits of data, the base unit of transfer in the RS232 configuration used here.

X indicates a bit in a don't care or undefined state
0 indicates a bit set to logic zero (cleared)
1 indicates a bit set to logic one (set)

Message structure within bytes is shown with a vertical separator bar. MSB (most significant bits) on the left, LSB (least significant bits) on the right.

Data direction : Data is **downloaded** to the master from the slave, and **uploaded** to the slave from the master. The slave devices are located ‘uphill’.

3 Serial Protocol - Timing

The timing of the serial data bits conforms to the RS232 industry standard for low data rate NRZ asynchronous communication. RS232 has various parameters and these are configured in the standard version as follows :

4800 baud, 1 start bit, 8 data bits, no parity bit, 1 stop bit

RS232 handshaking signals (RTS, CTS, DTR, DCD, DSR) are not supported. This is a small data packet control protocol and data overruns are not likely to occur.
There are 10 bits needed for each transmitted byte - 8 data bits + 1 start bit + 1 stop bit. No parity bit is used.

At 4800 baud, each byte takes **2.083 ms**.

There is an option to run at 9600 baud, in which case each byte takes **1.042 ms**.

3.1 Baud Rate

The 4800 baud rate was chosen as a lowest common denominator. Although many host devices can handle baud rates of 19,200 and above, some low power slave devices can only achieve their low power by running microcontrollers at low external clock speeds. This limits the maximum baud rate of internal baud rate generators. Even at a baud rate of 4800, a typical cctalk message only takes 30ms to 60ms. There is also the added benefit of noise immunity - it is requires a much higher level of noise to interfere with a bit lasting 208us compared to 52us.

It is possible to operate cctalk at the higher baud rate of **9600 baud**, but this should be made clear in any documentation accompanying the product. See appendix A5 for details of the cctalk naming convention.

4 Serial Protocol - Voltage Levels

A level-shifted version of RS232 is used for convenience and to reduce cost. This means there are no negative voltages about. On the serial connector, the idle state = +Vs nominal and the active state = 0V nominal.

Mark state (idle) **+Vs nominal**
Space state (active) **0V nominal**

Many products have been requested with 5V interfaces, in which case Vs = +5V.

The allowable voltage levels for each state are determined by the interface electronics and these may vary slightly from application to application. Further details are not given in this document but see Figures 1 & 2 for typical circuits.

5 The Serial Data Line

The transmit and receive messages take place on a single bi-directional serial DATA line. The other line is a 0V or COMMON line.

The standard cctalk interface is an open-collector NPN transistor driver on the DATA line with a pull-up resistor at the host end of the link. The value of the pull-up resistor will depend on the current-sinking ability of the communicating devices, the degree of noise immunity required and the maximum number of peripherals which can be attached to the bus. The ability to sink more current will result in better noise immunity.

There are no special screening requirements for short interconnection distances (less than 10m) since this is a low speed control network. Line drivers, opto isolators and twisted-pair cables are only likely to be necessary in the presence of high electrical noise. If cctalk is to be used over long interconnection distances (within or between rooms / departments) it is recommended that **RS485 drivers** are used rather than the open-collector interface.

5.1 RS485 Drivers

RS485 is a balanced transmission line system for use in noisy environments and over longer interconnection distances. It utilises an extra line for the serial data (balanced current) and requires a direction signal to control access to the multi-drop bus. PC-based software often uses the RTS handshaking signal with special driver software to toggle the direction status when sending out bytes.

A comparison of various electrical interfaces for serial communication is shown in the table below.

Interface	cctalk	RS232	RS485
Type	unbalanced multi-drop	point-to-point	balanced multi-drop
Data Lines	1	2	2
Direction Control	No	No	Yes
Max. Peripherals (Note α)	8	1	32
Max. Distance	50ft	50ft	4000ft
Max. Speed	9600	19.2K	10M
Mark (idle)	+5V	-5V to -15V	+1.5V to +5V (B > A)
Space (active)	0V	+5V to +15V	+1.5V to +5V (A > B)

α : Electrical limitation rather than protocol limitation.

6 Connector Details

The exact connector type is not a requirement of cctalk compatibility but obviously some kind of standardisation helps to reduce the number of cable converters in circulation. Different applications have different requirements and the choice of connector may be influenced by the product specification (e.g. robustness, humidity, power requirements…).
The following cctalk connector types are defined as follows :

Type	Pins	Description
1	4	standard interface, in-line connector
2	4	standard interface, locking connector
3	10	low power interface
4	6	extended interface, in-line connector
5	10	AWP industry-standard interface

Type 1 (standard interface, in-line connector)

(1) +Vs
(2) <key>
(3) 0V
(4) /DATA

Recommended peripheral connector :
Molex 42375 Series 0.1inch pitch straight flat pin header
P/N 22-28-4043 (15µ gold)

Mates with :
Molex 70066 Series single row crimp connector housing
P/N 50-57-9304
(Alternative : Methode 0.1inch IDC connector 1308-204-422)

Crimps :
Molex 70058 Series
P/N 16-02-0086 (15µ gold)

Type 2 (standard interface, locking connector)

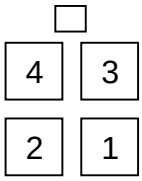
(1) +Vs
(2) <reserved>
(3) 0V
(4) /DATA

Recommended peripheral connector :
Molex 3.00mm pitch Micro-Fit 3.0 Wire-to-Board Header (vertical mounting)
P/N 43045-0413 (15µ gold)

Mates with :
Molex 3.00mm pitch Micro-Fit 3.0 Wire-to-Wire Receptacle P/N 43025-0400

Crimps :
P/N 43030-0002 (15µ gold)

Pin Polarity :



View of socket from front.

Type 3 (low power interface)

(1) /DATA
(2) 0V (shield)
(3) /REQUEST POLL
(4) 0V (shield)
(5) /RESET
(6) <key>
(7) /INHIBIT ALL
(8) 0V (logic)
(9) +5V
(10) 0V (solenoid)

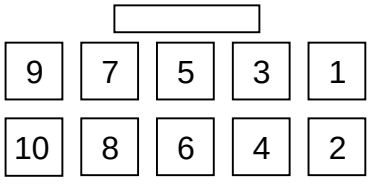
Recommended peripheral connector :
Molex 8624 Series 0.1inch dual row straight pin breakaway header
P/N 10-89-1101 (15µ gold)

Mechanical keying should be provided by the surrounding cover.

(Alternative :
Molex 70246 Series dual row straight pin low profile shrouded header 70246-1021)

Mates with :
Molex 40312 Series MX50 ribbon cable connector system
P/N 15-29-9710 (15µ gold, centre polarisation, strain relief)

Pin Polarity :



View of pin header from front.

Type 4 (extended interface, in-line connector)

(1) +Vs
(2) <key>
(3) 0V
(4) /DATA
(5) /RESET
(6) /REQUEST POLL

Recommended peripheral connector :
See type 1 connector

Type 5 (AWP industry-standard interface)

In the UK, this connector is specified by BACTA for use in all AWP machines with serial coin acceptors.

This type of connector supports both Mars HI^2 (Host Intelligent Interface © Mars Electronics International) and MCL cctalk protocols.

(1) /DATA	← cctalk interface	
(2) DATA 0V		internally connected to 0V
(3) /BUSY		not used in cctalk (not connected)
(4) BUSY 0V		internally connected to 0V
(5) /RESET		optional use in cctalk
(6) /PF		not used in cctalk (not connected)
(7) +12V	← cctalk interface	
(8) 0V	← cctalk interface	
(9) /SERIAL MODE	← cctalk interface, **connect to 0V for serial opera**tion	
(10) +12V alt.		not used in cctalk (not connected)

cctalk does not require as many signals as HI^2.
For coin acceptors which have both serial and parallel interfaces, the /SERIAL MODE signal is used to indicate serial operation rather than parallel operation.

To see which serial protocols are supported by the coin acceptor, it is suggested that a test message is sent out in one of the protocols and the reply message (if any) checked. In cctalk, a suitable first message is the ‘Simple poll’ command.

See connector Type 3 for more details.

7 Interface Details

See Figure 1 for the standard cctalk interface circuit.
See Figure 2 for a cut-down version.

Note that both these circuits are for +5V operation only.

8 Message Structure

Standard Message Packet (header byte, N data bytes)

[Destination Address]
[No. of Data Bytes]
[Source Address]
[Header]
[Data 1]
...
[Data N]
[Checksum]

Each communication sequence (a command or request for information) consists of 2 message packets structured in the above manner. The first will go from the master device to a slave device and then a reply will be sent from the slave device to the master device. The reply packet could be anything from a simple acknowledge message to a stream of data. **Note that the acknowledge message conforms to the above structure in the same way all other messages do.**

The structure does not alter according to the direction of the message packet. The serial protocol structure *does not care* who originates the message and who responds to it.

If there are no data bytes then the message format is shortened as follows :

Short Message Packet (header byte, no data bytes)
[Destination Address]
[0]
[Source Address]
[Header]
[Checksum]

Note : [0] indicates that there are zero data bytes in the message.

8.1 Destination Address

Range 0 to 255 (254 slave addresses)

0 : broadcast message (see next heading)
1 : the usual host (master) address
2 : the usual slave address for non multi-drop networks
3 to 255 : various slave addresses for multi-drop networks

8.1.1 The Broadcast Message

A destination address of '0' is a special case whereby all attached devices respond. In this case the returned source address is '0' indicating a *reply by all*.
This command should be used with caution in a multi-drop network as all the attached devices will send replies that collide with each other - although this can be allowed for. Refer to the MDCES section for commands which can safely make use of a broadcast message.

8.2 No. of Data Bytes

Range 0 to 252.

This indicates the number of **data bytes** in the message, not the total number of bytes in the message packet. A value of '0' means there are no data bytes in the message and the total length of the message packet will be 5 bytes - the minimum allowed. A value of 252 means that another 255 bytes are to follow, 252 of which are data.

Although it would be theoretically possible to have 255 data bytes, implementation is helped in small devices such as PIC microcontrollers by having no more than 255 bytes following the 'No. of Data bytes'. Allowing for the source address, header and checksum, this gives the value 252.

The values 253 to 255 are illegal and should be treated by the slave as if the value was 252. Likewise the checksum will be calculated for a maximum of 252 data bytes.

8.2.1 Long Transfers

In some circumstances there will be a requirement to transfer more than 252 bytes of data. This is achieved by splitting the data into blocks, 1 for each message, and issuing a sequence of commands. The block size may be the maximum of 252 bytes or it may be more convenient to transfer blocks of 128 bytes.

This approach is good from the data integrity point of view as the checksum is better able to detect errors.

A good example of block transfer is the 'Read data block' command and the 'Write data block' command.

8.3 Source Address

Range 1 to 255.

The value of '0' is avoided as the response packet would be broadcast to all attached devices - undesirable in a multi-drop network !

8.4 Header

Range 0 to 255.

Header bytes have been defined to cover a broad range of activities in coin validation, cash handling and other electromechanical equipment. Header bytes will be kept the same across a broad range of products although the exact meaning and interpretation of data bytes could vary.
The header value of '0' indicates a response packet. A slave device should not be sent a null header by the master, it should only return one.

8.5 Data

Range 0 to 255.

No restrictions on use. The data may have various formats such as binary, BCD and ASCII.

8.6 Checksum

This is a simple zero checksum such that the 8-bit addition (modulus 256) of all the bytes in the message from the start to the checksum itself is zero. If a message is received and the addition of all the bytes is non-zero then an error has occurred. See 'Action on Error' heading.

For example, the message [1] [0] [2] [0] would be followed by the checksum [253] because 1 + 0 + 2 + 0 + 253 = 256 = 0.

8.6.1 Checksum Theory

Most serial protocols in the coin / vending industry are protected with either simple 8-bit addition checksums or 16-bit checksums. The algorithms for 16-bit checksums vary from addition to cyclic redundancy codes such as CRC-CCITT and CRC-16 with their mathematical basis in polynomial division (the idea being that division is far more sensitive to bit errors than addition). The processing power required for CRC calculations is greater than addition checksums, but where code space is plentiful, a 512 byte look-up table can improve performance to near that of addition.

The abilities of the various checksum algorithms to detect errors are summarised below :

8-bit checksum
Can detect all single-bit errors.
Can detect most double-bit errors.

This method is not recommended for physical links noisy enough to give a significant probability of more than one corrupted bit per message.

16-bit checksum (addition)
Much better than the 8-bit checksum but still not capable of detecting all double-bit errors.

CRC-CCITT / CRC-16
Can detect all single-bit errors.
Can detect all double-bit errors.
Can detect all odd numbers of bit errors

All methods are good at detecting burst errors - blocks of zeros or ones.
The high-level structure of the cctalk protocol means that for most situations an 8-bit checksum is perfectly adequate. There is some redundancy in the returned message packets which helps error detection. For instance, errors in the destination and source addresses are easy to spot and errors in the number of data bytes can result in a timeout or overrun.

9 The Acknowledge Message

If a message has a null header (which is the case for a reply packet) and no data bytes then this is considered a simple slave acknowledge message.

[Destination Address]
[0]
[Source Address]
[0]
[Checksum]

9.1 The NAK Message

The NAK message is not normally used on cctalk systems but is provided for completeness.

The NAK message has limited use in a cctalk multi-drop network because an incorrectly received message cannot be assumed to have a correct source or destination address. Replying to non-existent or incorrectly addressed hosts merely clogs up valuable bandwidth. However, if an operation carried out by the slave device (after receiving an error-free command) fails, then a NAK message may be appropriate. The way most cctalk commands work is to include status information regarding the success of the command in the returned data as this can be tailored very specifically to the action performed and is of more use to a host controller than a generic ‘NAK’ message.

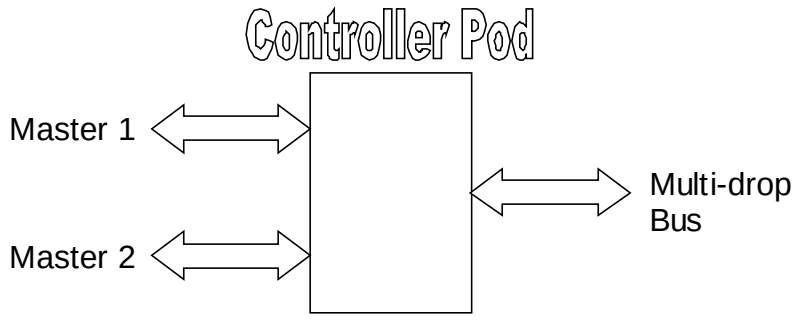
The NAK message is defined as :

[Destination Address]
[0]
[Source Address]
[5]
[Checksum]

Note that in this case the return header is non-zero.

9.2 The BUSY Message

The BUSY message is not normally used on cctalk systems but is provided for completeness.



Due to the complexity of multi-master systems, it can be preferable to use a multi-master controller pod sitting on the cctalk bus to arbitrate between 2 or more master devices. The controller pod can assign priorities to the various serial commands and deny one master access to the bus when another is waiting for a reply. For this purpose, a BUSY return message has been defined. If a master receives a BUSY response, it must retry after a (preferably random) amount of time.

The BUSY message is defined as :

[Destination Address]
[0]
[Source Address]
[6]
[Checksum]

Note that in this case the return header is non-zero.

10 Commands that return ASCII Strings

By convention, ASCII strings are returned in the same order as they print or appear on screen. For instance, "Hello" is returned as...

['H']
['e']
['l']
['l']
['o']

The length of the string can be determined from the [No. of Data Bytes] byte sent as part of the message packet. There is no *null terminator* as in a 'C' language character string.

No. of Data Bytes = length of string

11 Implementation Details

A bi-directional serial data line can be tackled by the design engineer in a variety of ways. A simple microcontroller implementation may use a single bi-directional I/O port and switch between an output for transmission and an input for reception. cctalk is a half-duplex protocol and so this approach should not present any problems. The slave device will not return any data until the entire transmit message is complete.

A microcontroller which has a built-in UART may have separate transmit and receive data lines. These can be combined in the interface circuitry. The software for a UART implementation will have to deal with transmitted data being immediately received again (loop-back). This can be tackled in software by using a receive enable / disable flag or by receiving the transmit message in full and then *ignoring it*.

Since the cctalk protocol does not use a wake-up bit which is a key feature of other serial protocols, all receive messages should be processed. This is a quick operation in a slave device and it should not affect the performance of the core software.

A brief outline of the receive algorithm is :

Get destination address

Get no. of data bytes

If { address match }

receive and store rest of message (count incoming bytes)
validate checksum
< execute command >

Else

receive rest of message (count incoming bytes)

End If

Receive timeout error : < clear receive counters >

12 Timing Requirements

The timing requirements of cctalk are not very critical but here are some guidelines.

Between bytes

When receiving bytes within a message packet, the communication software should wait up to **50ms** for another byte if it is expected. If a timeout condition occurs, the software should reset all communication variables and be ready to receive the next message. No other action should be taken.
The ability to transparently time-out and respond to the next valid command is a key feature of the robustness of cctalk and must be adhered to.

Between command and reply

The delay between the issuing of a command and the reply being received is command dependent and application specific. Some commands return data immediately (within 10ms) while others wait for some action to be performed first. For instance, a command that pulses a solenoid will only send an ACK message when the solenoid has finished activating. This could be a couple of seconds later. Host software should be written to take account of these variable delays and time-out according to the command sent. To test whether a slave device is operational, choose a command with a fast response time - the 'Simple poll' command is an ideal choice. Well-written slave devices should respond as soon as possible to minimise network latency.

13 Action on Error

A host device will transmit a message to a slave device. If the slave device detects an error condition such as a bad checksum or missing data (receive timeout) then no further action is taken and the receive buffer is cleared. The host device, on receiving no return message, has the option of re-sending the command. Likewise, if the host does not receive the return message correctly, it has the option of re-sending the command. **It is up to the host device whether re-sending the command occurs immediately or after a fixed or random delay.**

A key feature of this protocol is that there are no NAK messages issued on detection of a receive error condition. This simplifies multi-drop implementations (if the message contents are not valid then who sent it ?) and reduces the chance of bus collisions. **The host device must decide what to do if there is no reply to a command.**

13.1 Retransmission

Some protocols have retransmission capability built into the transport layer (see Appendix A1). The philosophy of cctalk is to keep the protocol implementation as simple as possible by having retransmission performed at a higher level. Most commands can have a very simple retransmission algorithm - if the checksum is wrong then the command is re-sent. Factors such as how many times retransmission is attempted before giving up can be varied to suit the application rather than being rigidly enforced by the protocol. Commands such as paying out coins from a hopper need a secure algorithm and this has been taken care of in the high level command structure.

Retransmission is implemented in cctalk at a high level and varies according to the application requirements.

14 Unrecognised Headers

If a slave device does not recognise (i.e. support) a particular header then no information is returned to the host device and no action is taken.

15 Practical Limitations in Low Cost Slave Devices

A typical low cost slave device has restrictions on ROM space, RAM space and processing capability. Therefore, the following limitations may apply on the slave device.

- The receive buffer is relatively small (anything from 1 to 10 bytes) and so very long messages from the host device cannot be received and stored. The slave will usually store data until the receive buffer is full.

- Although there is support in the protocol for variable length messages, the slave device does not have the power or flexibility to deal with them. The slave will assume a particular header has a certain number of data bytes.

16 Commands

16.1 Implementation Levels

The cctalk implementation level is a number (0 to 255) indicating the degree of support for cctalk serial commands. In other words, the number tells the host software which command headers are replied to.

It is envisaged that the implementation level will be used with respect to a particular model of peripheral device rather than globally. Therefore, if a particular model is upgraded by adding support for a few extra serial commands, the implementation level number will be incremented so that the host machine can use both old and new product. It is a requirement that all future implementation levels are backwardly compatible such that responses to existing commands are unchanged.

Every product will be provided with a table showing which command headers are supported against the various implementation levels. New products will only have one implementation level.

Implementation levels will start at 1 and grow 2, 3, 4…

Note : the complex definition of implementation level as described in version 2.0 of the specification has been abandoned.

To read the implementation level electronically, see command header 4.

16.2 Commands in Detail

The command descriptions below form the basis of a generic specification. It is often necessary to obtain more precise details on how the commands operate with respect to a specific product. In this case reference should be made to, as it is called below, the *product manual*. This will be some kind of product specification or functional specification which describes the action of various serial commands in more detail.

16.2.1 Command Description Notation

Transmitted and received data are described with respect to the **host**.

The slave returns an ACK even if there is no return data. This way the host knows that the command was received correctly and it is safe to send the next one.

Received data : <none> ← indicates an ACK message only

16.2.2 Command List

Header 255 - Factory set-up and test

This command is reserved for functions needed during the manufacture of a product and is not intended for general use.

Header 254 - Simple poll

Transmitted data : <none>
Received data : <none>

This command can be used to check that the slave device is powered-up and working. No data is returned (ACK only) and no action is performed. It can be used at EMS (Early Morning Start-up) to check that the slave device is communicating. A timeout on this command indicates a faulty or missing device.

*** **All cctalk peripherals must reply to a poll** ***

Header 253 - Address poll to determine attached devices
Refer to Appendix A

Header 252 - Address clash to determine if any devices have the same address
Refer to Appendix A

Header 251 - Address change to change the address of the attached device
Refer to Appendix A

Header 250 - Address random to randomly change the address of a device
Refer to Appendix A

Header 249 - Request polling priority

Transmitted data : <none>
Received data : [units] [value]

This is an indication by the slave of how often it should be polled. This is very application specific but given the same application, slave devices with lots of memory may have internal buffers to reduce the polling requirement.

[units]
0 - special case
1 - ms
2 - 10 ms
3 - seconds
4 - minutes
5 - hours
6 - days
7 - weeks
8 - months
9 - years

If units = 0 and value = 0 then refer to product manual for polling information
If units = 0 and value = 255 then the device uses a hardware REQUEST POLL line

Header 248 - Request status

Transmitted data : <none>
Received data : [status]

This command reports the status of the slave device.

Refer to Table 5.

Header 247 - Request variable set

Transmitted data : <none>
Received data : [variable 1] [variable 2] [variable 3] ...

This command requests unspecified variables from the slave device. Some of the returned variables may be useful to a host application and some may not, but any data is application specific. Refer to the product manual for more details.

Header 246 - Request manufacturer id

Transmitted data : <none>
Received data : <ASCII string>

The manufacturer's id string is returned.

Header 245 - Request equipment category id

Transmitted data : <none>
Received data : <ASCII string>

The equipment category id string is returned.

Refer to table 2 for a list of standard category strings.

Header 244 - Request product code

Transmitted data : <none>
Received data : <ASCII string>

The product code is returned. No restriction on format.

Header 242 - Request serial number

Transmitted data : <none>
Received data : [serial 1] [serial 2] [serial 3]

This command returns the device serial number in binary and for most products a 3 byte code is sufficient.

24-bit serial number in decimal = [serial 1] + 256 * [serial 2] + 65536 * [serial 3]
Range 0 to 16,777,215 (~16 million)

Adding an extra data byte will increase the largest product serial number to…
4,294,967,295 (~4 billion).

Header 241 - Request software revision

Transmitted data : <none>
Received data : <ASCII string>

The slave device software revision is returned. There is no restriction on format - it may include non-numeric characters.

Header 240 - Test solenoid

Transmitted data : [bit mask]
Received data : <none>

The device solenoids are pulsed.
The length of the pulse is product specific. The slave ACK is returned **after** the solenoid de-activates.

[bit mask]
The following bits have been defined for a coin acceptor :
Bit 0 - Accept gate solenoid
0 = no action 1 = pulse

Header 238 - Test output line

Transmitted data : [bit mask]
Received data : <none>

The length of the pulse is product specific. The slave ACK is returned **after** the solenoid de-activates.
[bit mask]
0 = no action
1 = pulse

Header 237 - Read input lines

Transmitted data : <none>
Received data : [data 1] [data 2] [data 3] ... refer to product manual

This command requests input data from the slave device. It can be used to check the operation of switches, push buttons, connector signals etc.

Header 236 - Read opto states

Transmitted data : <none>
Received data : [opto states]

This command is used to check the state of various opto devices in the slave device. Refer to the product manual for more details.

[opto states]
0 = opto clear
1 = opto blocked

Header 233 - Latch output line

Transmitted data : [bit mask]
Received data : <none>

The output line is latched. Polarities will be detailed in the product manual.

Header 232 - Perform self-check

Transmitted data : <none>
Received data 1 : [fault code]
Received data 2 : [fault code] [extra info]

This command instructs the slave device to perform a self-check. The actual level of testing is decided by the slave rather than the host and a fault code is returned. Some slave devices support an additional byte of information for certain fault codes.

[fault code]

Refer to Table 4.
Header 231 - Modify inhibit status

Transmitted data : [inhibit mask 1] [inhibit mask 2]
Received data : <none>

This command sends an individual coin inhibit pattern to a coin acceptor.

[inhibit mask 1]
Bit 0 - coin 1
...
Bit 7 - coin 8

[inhibit mask 2]
Bit 0 - coin 9
...
Bit 7 - coin 16

0 = coin disabled (inhibited)
1 = coin enabled

The product manual should make clear whether these changes are permanent (stored in non-volatile memory) or temporary (stored in RAM).

Header 230 - Request inhibit status

Transmitted data : <none>
Received data : [inhibit mask 1] [inhibit mask 2]

This command requests an individual coin inhibit pattern from a coin acceptor.

[inhibit mask 1]
Bit 0 - coin 1
...
Bit 7 - coin 8

[inhibit mask 2]
Bit 0 - coin 9
...
Bit 7 - coin 16

0 = coin disabled (inhibited)
1 = coin enabled

Header 229 - Read buffered credit or error codes

Transmitted data : <none>
Received data : [events] [result 1A] [result 1B] ...

This command is similar to the 'Read last credit or error code' but a past history of event codes is returned rather than the last one. This enables a host device to poll the slave at a much lower frequency while not missing any credits.
The number of event codes that can be *stacked* by the slave device is product specific - refer to the product manual.
A new event ripples data through the return data buffer and the oldest event is lost.
For example, consider a 5 event buffer :

result 5A ⇨ lost
result 5B ⇨ lost

result 4A ⇨ result 5A
result 4B ⇨ result 5B

result 3A ⇨ result 4A
result 3B ⇨ result 4B

result 2A ⇨ result 3A
result 2B ⇨ result 3B

result 1A ⇨ result 2A
result 1B ⇨ result 2B

new result A ⇨ result 1A
new result B ⇨ result 1B

[events]
0 (power-up or reset condition)
1 to 255 - event counter

The event counter is incremented every time a new credit or error is added to the buffer. When the event counter is at 255 the next event causes the counter to change to 1. The only way for the counter to be 0 is at power-up or reset.

By examining the event counter between polls, the host device can determine how many **new** events have occurred. For example, if the event counter has not changed then no new events have occurred.

Examples

last event counter	current event counter	new events
23	23	0
102	104	2
255	1	1
253	3	5
67	0	unknown - power fail

[result xA] [result xB]
3 format types have been defined for coin acceptors :

[coin position] [0]
[coin position][sorter path]
[0] [error code]

For more information, refer to the 'Read last credit or error code' command.
Reduced Form

Some products may only support a single byte code in the event buffer. In this case the [result xB] codes are not used.

Refer to the product manual for details of what the single bytes refer to.

e.g. [coin position] or [binary accept code]

Header 226 - Request insertion counter

Transmitted data : <none>
Received data : [count 1] [count 2] [count 3]

This command returns the total number of coins put through a coin acceptor.

24-bit counter in decimal = [count 1] + 256 * [count 2] + 65536 * [count 3]
Range 0 to 16,777,215

Restrictions on use and likely accuracy will be made clear in the product manual.

Header 225 - Request accept counter

Transmitted data : <none>
Received data : [count 1] [count 2] [count 3]

This command returns the number of coin accepts made by a coin acceptor.

24-bit counter in decimal = [count 1] + 256 * [count 2] + 65536 * [count 3]
Range 0 to 16,777,215

Restrictions on use and likely accuracy will be made clear in the product manual.

Header 216 - Request data storage availability

Transmitted data : <none>
Received data : [memory type] [read blocks] [read bytes per block]
[write blocks] [write bytes per block]

Some slave devices allow host data to be stored for whatever reason. Whether this service is provided at all can be determined by using this command.

[memory type]
0 - volatile (lost on reset)
1 - volatile (lost on power-down)
2 - permanent (limited use)
3 - permanent (unlimited use)

Memory types 0 and 1 are typically implemented in RAM, type 2 in EEPROM (write life cycle between 10K and 10M) and type 3 in battery-backed RAM.

[read blocks]
0 (256 blocks of read data available)
1 to 255 - number of blocks of read data available

[read bytes]
0 (no read data service)
1 to 252 - number of bytes per block of read data

[write blocks]
0 (256 blocks of write data available)
1 to 255 - number of blocks of write data available

[writes bytes]
0 (no write data service)
1 to 251 - number of bytes per block of write data

Due to variable packet length restrictions in the base protocol, we arrive at the following capacities :

The minimum read capacity is 1 block of 1 byte = 1 byte
The maximum read capacity is 256 blocks of 252 bytes = 64,512 bytes
(98.4% of 64K)

The minimum write capacity is 1 block of 1 byte = 1 byte
The maximum write capacity is 256 blocks of 251 bytes = 64,256 bytes
(98.0% of 64K)

Note that the format of the read blocks may be different to the format of the write blocks, but they should nevertheless be contiguous locations in memory. Since write cycle times (the slave will embody automatic erase cycles if necessary) are usually much longer than read cycle times in permanent memory types, it is sometimes better to read data in one large chunk but write it in smaller chunks. This ensures command response times are better and multi-drop polling can be *finer grained*.

Header 213 - Request option flags

Transmitted data : <none>
Received data : [option flags]

This command reads option flags (single bit control variables) from the slave device.
[option flags]
Bit 0 - serial credit codes represent…
0 = coin position
1 = relative coin value (see section 21 on CVF)
Bits 1 to 7 - <undefined>

Header 212 - Request coin position

Transmitted data : [value]
Received data : [position mask 1] [position mask 2]

This command can be used in coin acceptors to locate the bit position of a given coin. The bit position ties up with the 'Modify inhibit status' command for inhibiting individual coins.

[value]
The coin credit code (coin position) or value as returned by the ‘Read buffered credit or error codes’ command.

Header 210 - Modify sorter paths

Transmitted data 1 : [coin position] [path]
Transmitted data 2 : [coin position] [path 1] [path 2] [path 3] [path 4]
Received data : <none>

This command allows sorter paths to be modified in a coin acceptor. Some coins have override paths and these are sent along with the primary path (path 1).

[coin position]
e.g. 1 to 6, 1 to 12, 1 to 16, 1 to 32

[sorter path]
e.g. 1 to 2, 1 to 4, 1 to 8

The product manual should make clear whether this change is permanent (stored in non-volatile memory) or temporary (stored in RAM).

Header 209 - Request sorter paths

Transmitted data : [coin position]
Received data 1 : [path]
Received data 2 : [path 1] [path 2] [path 3] [path 4]

This command allows sorter paths to be requested in a coin acceptor. Some coins have override paths and these are sent along with the primary path (path 1).

[coin position] e.g. 1 to 6, 1 to 12, 1 to 16, 1 to 32

[sorter path] e.g. 1 to 2, 1 to 4, 1 to 8

Header 202 - Teach mode control

Transmitted data : [coin position]
Received data : <none>

[coin position]
e.g. 1 to 12, 1 to 16

Puts coin acceptor into 'teach' mode.

Header 201 - Request teach status

Transmitted data : [mode]
Received data : [no. of coins entered] [status code]

[mode]
0 - default
1 - abort validator teach mode

[status code]
252 - teach aborted
253 - teach error
254 - teaching in progress (busy)
255 - teach completed

Header 197 - Calculate ROM checksum

Transmitted data : <none>
Received data : [checksum 1] [checksum 2] [checksum 3] [checksum 4]

The method of calculating the ROM checksum is not defined in this document and can be adapted to the slave device. A simple 'unsigned long' addition can be assumed to be the most common method, displayed as 8 hex digits.

Header 196 - Request creation date

Transmitted data : <none>
Received data : [date code LSB] [date code MSB]

[date code]
Bits 0 to 4 - Day (1 to 31)
Bits 5 to 8 - Month (1 to 12)
Bits 9 to 13 - Year (PRODUCT_BASE_YEAR + 0 to 31)
Bits 14 to 15 - Reserved

This command returns the factory creation date of the product.

The PRODUCT_BASE_YEAR is defined in the product manual. Since the date storage is relative, there are no 'millennium bug' issues.

Header 195 - Request last modification date

Transmitted data : <none>
Received data : [date code LSB] [date code MSB]

[date code]
Bits 0 to 4 - Day (1 to 31)
Bits 5 to 8 - Month (1 to 12)
Bits 9 to 13 - Year (PRODUCT_BASE_YEAR + 0 to 31)
Bits 14 to 15 - Reserved

This command returns the last modification date of the product. It is usually changed by remote programming toolkits or PC-based support software.

The PRODUCT_BASE_YEAR is defined in the product manual. Since the date storage is relative, there are no 'millennium bug' issues.

Header 194 - Request reject counter

Transmitted data : <none>
Received data : [count 1] [count 2] [count 3]

This command returns the number of reject coins put through a coin acceptor.

24-bit counter in decimal = [count 1] + 256 * [count 2] + 65536 * [count 3]
Range 0 to 16,777,215

Restrictions on use and likely accuracy will be made clear in the product manual.

Header 193 - Request fraud counter

Transmitted data : <none>
Received data : [count 1] [count 2] [count 3]

This command returns the number of pre-programmed fraud coins put through a coin acceptor and can be used to monitor a fraud attack on a particular site.

24-bit counter in decimal = [count 1] + 256 * [count 2] + 65536 * [count 3]
Range 0 to 16,777,215

Restrictions on use and likely accuracy will be made clear in the product manual.

Header 192 - Request build code

Transmitted data : <none>
Received data : <ASCII string>

The product build code is returned. No restriction on format.

The complete product id can be determined by using 'Request product code' followed by 'Request build code'.

Header 185 - Modify coin id

Transmitted data : [coin position] [char 1] [char 2] [char 3]…
Received data : <none>

[coin position]
e.g. 1 to 6, 1 to 12, 1 to 16, 1 to 32

Some coin acceptors can store an identification string alongside the normal validation parameters for each coin. Refer to the product manual for details. The identification string typically consists of 6 ASCII characters.

Header 184 - Request coin id

Transmitted data : [coin position]
Received data : [char 1] [char 2] [char 3]…

[coin position]
e.g. 1 to 6, 1 to 12, 1 to 16, 1 to 32

Header 183 - Upload window data

Transmitted data 1 : [mode] [coin position]
[data 1] [data 2] [data 3]…
Transmitted data 2 : [mode] [coin position] [credit code]
Transmitted data 3 : [mode] [coin position]
Received data : <none>

[mode]
0 - program coin window
1 - program credit code
2 - delete coin window

This command allows remote programming of coin windows in the event that the ‘Upload coin data’ command is not supported.

Header 4 - Request comms revision

Transmitted data : <none>
Received data : [cctalk level] [major revision] [minor revision]

This command requests the cctalk level number and the major / minor revision numbers of the comms specification. This is read separately to the main software revision number for the product which can be obtained with a 'Request software revision' command.

The revision numbers should tie up with those at the top of this specification document.

Header 3 - Clear comms status variables

Transmitted data : <none>
Received data : <none>

This command clears the comms status variables (cumulative single byte event counters). See 'Request comms status variables' command for more details.

Header 2 - Request comms status variables

Transmitted data : <none>
Received data : [rx timeouts] [rx bytes ignored] [rx bad checksums]

There are 3 cumulative single byte event counters (the value 255 wraps around to 0) that can be requested with this command. For test purposes, the counters should be cleared first with the 'Clear comms status variables' command.

[rx timeouts] - cumulative event counter
This is the number of times the slave device has timed out on receiving data. This should be zero for a good communication link and a correctly implemented communication algorithm. The slave device should count and ignore incorrectly addressed messages and unrecognised headers without timing out. It should also handle the MDCES commands without registering a timeout. Since the length of all messages is contained within the message structure or is known in advance then this should not be a problem.

[rx bytes ignored] - cumulative event counter
If a long message is sent to the slave device, not all of which can be stored in the receive buffer, then the number of bytes lost is added to this counter. Note that the slave device should still be able to calculate the checksum even if some of the receive data is not stored.

[rx bad checksums] - cumulative event counter
This counter is incremented each time a receive message is obtained with an incorrect checksum and a valid slave address. This variable can be monitored for signs of a noisy communication link.

Header 1 - Reset device

Transmitted data : <none>
Received data : <none>

This command forces a *soft* reset in the slave device. It is up to the slave device what action is taken on receipt of this command and whether any internal house-keeping is done. The action may range from a jump to the reset vector to a forced physical reset of the processor and peripheral devices. This command is worth trying before a hard reset (or power-down) is performed. Note that on reset, any PIN number settings may be cleared and will need to be re-entered.

17 Implementing cctalk on a New Product

Although originally designed for use in the coin industry, there is no reason why cctalk cannot be used in other areas. Choose existing header numbers if possible to implement the function required. For instance, a lot of the function headers have generic capability such as 'Read input lines', 'Test output lines', 'Test solenoids' and 'Modify master inhibit status'. The exact implementation of the command parameters can be documented with the product.

If there are some functions which are totally unlike anything described in the header definition table then a new header code will be needed. Header numbers 7 to 99 are reserved for this purpose.

17.1 Polling Algorithms

In a typical coin handling application, only 2 commands need to be sent regularly by the host machine. These are ‘Read buffered credit or error codes’ and ‘Request payout status’. Then again, ‘Request payout status’ only needs to be used when paying out coins from a hopper.

This is a quick guide to the basic software needed for cctalk polling :

Initialisation

Issue ‘Read buffered credit or error codes’ and store event counter
Issue ‘Request payout status’ and store event counter

Continuous Host Polling

Issue ‘Read buffered credit or error codes’ and check event counter
If checksum bad or general comms error then retry

If events = 0 and last events > 0 then error condition (power fail and possible lost credits)
If delta(events) = 0 then no new credits
If delta(events) >= 1 and delta(events) <= 5 then new credit information
If delta(events) > 5 then error condition (1 or more lost credits)

If payout command in progress …
    Issue ‘Request payout status’
    If checksum bad or general comms error then retry

    If events = last events then re-send ‘Dispense coins’
    If events = 0 then error condition (power fail, payout terminated)
    If events = last events + 1 then check status code

    If status = ‘paying out’ then continue polling
    If status = ‘error’ then error condition
    If status = ‘payout completed’ then finished

18 Writing Generic Host Software Applications

It is possible with care to write generic host software to deal with any standard cctalk peripheral connected to it. Doing this obviously requires a lot more effort than writing a host application for one specific peripheral type. A core set of cctalk commands should be run through first to discover what type of peripheral it is.

Command	Return Data Type	Information Gained
Request equipment category id	ASCII	What sort of peripheral is it ?
Request comms revision	Binary	What kind of cctalk is it ?
Request manufacturer id	ASCII	Who manufactures it ?
Request product code	ASCII	Which model is it ?
Request build code	ASCII	Which build is it ?
Request software revision	ASCII	Which revision is it ?

Armed with this information the host application can consult a look-up table of commands and use the appropriate ones for the peripheral concerned. Clearly, the more standardisation there is in the industry over the data format for different commands, the less complex the look-up table needs to be.

19 Multi-Drop Considerations

A multi-drop environment is obviously a lot more complicated than a simple interconnection between one master device and one slave device. It could potentially be a multi-master, multi-drop bus where several devices could all be trying to communicate at once. This could result in *clashes* or collisions of data packets and in this implementation of the serial protocol the only way to detect a bus collision is the checksum byte. There might additionally be a RS232 framing error detected by the UART which means the stop bit was not in the expected place (the 10th and last bit should always be high). Although it would be possible to implement a multi-master system in theory using low data rates and backing-off for a random amount of time in the event of a collision, cctalk is not the best choice of protocol for multi-master systems. We will assume here the multi-drop application of interest is a single-master, multi-slave network.

The most likely mode of operation is where the master device polls through the slave devices in turn, requesting status information and transferring data when necessary. This avoids any possibility of bus contention but requires polling times for each peripheral device to be within specification. In cash handling applications, a polling time of between 100ms and 500ms is usually perfectly adequate.

The cctalk 2-wire serial interface has been designed to allow a number of peripherals to be chained together. There is no restriction on connection topology - they can be linked in-line, in a ring or in a tree structure.

Note that there is a common 0V line running between all peripheral devices - if this is a problem due to differing ground potentials then special circuitry needs to be used - for instance the use of opto isolators.

The maximum length of connection will depend on the degree of electrical noise (radiated and conducted) in the system. We are operating at very low baud rates, which makes matters better, but

at low voltages, which makes matters worse. In its simplest form, cctalk is not designed to operate over more than 10 metres of unscreened cable.

19.1 MDCES - Multi-Drop Command Extension Set

The **cctalk MDCES** (multi-drop command extension set) gives additional functionality to multi-drop applications. However, some MDCES commands by necessity do not conform to the standard cctalk message format.

In some situations it is possible to have device addresses configured before use so that no address ambiguities arise. In other situations, new devices will be plugged into the network without prior knowledge. The host controller can then perform a network scan to automatically determine new addresses and resolve address ambiguities. It is in these circumstances the MDCES becomes useful. Note that the current serial protocol does not support *hot-plugging* of peripherals - the host device must be informed by other means of a change in network configuration.

The key problem of a multi-drop network is all like-addressed devices responding at once. The response of 2 special commands has been limited to 1 byte to reduce the collision complexities. These commands are the 'Address poll to determine attached devices' command and the 'Address clash to determine if any devices have the same address' command. The byte they return is delayed by a certain amount. The address poll command delays the response by a time proportional to the address value. The address clash command delays the response by a time proportional to a random value. After sending one of these two commands, some or all of the following events could occur :

a) **No collision**. The returned bytes are completely separate.
b) **Collision**. The returned bytes overlap but are staggered in time i.e. the start bits are non-synchronous. This can result in a framing error or 1 or more bytes with the wrong value.
c) **Unison**. The returned bytes are in unison and are read as a single byte with no errors. Although in theory 2 devices with the same address should respond identically in time, variations in clock speeds, asynchronous internal timing and physical propagation delays means this is unlikely to be the case.

Although the possibilities look complicated, intelligent host software making use of the commands detailed below can sort the mess out. Obviously there is no problem for a pre-configured network with all address ambiguities resolved beforehand.

Header 253 and header 252 are used to check that there are no address ambiguities.

The most important command of all is header 250 which can randomise a slave device address. This can be done on a subset of slaves with the same destination address or across the entire network (destination address = 0).

Header 253 - Address poll to determine attached devices
↳ Special Format

The host issues this command with a zero destination address so that all attached devices respond.

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

This command is used to determine which devices are connected to the bus by requesting that all attached devices return their address. To avoid collisions, only the address byte is returned and it is returned at a time proportional to the address value.

Slave Response Algorithm

Disable rx port
Delay (4 * addr) ms
Send [addr]
Delay 1200 - (4 * addr) ms
Enable rx port

If the host device receives all bytes returned for about **1.5s** after issuing this command, it can determine the number and addresses of attached devices.

Header 252 - Address clash to determine if any devices have the same address
↳ Special Format

The host issues this command with a specific destination address.

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

This command is used to determine if one or more devices share the same address. To avoid collisions, only the address byte is returned and it is returned at a time proportional to a random value between 0 and 255.

Slave Response Algorithm

r = rand(256)
Disable rx port
Delay (4 * r) ms
Send [addr]
Delay 1200 - (4 * r) ms
Enable rx port

If the host device receives all bytes returned for about **1.5s** after issuing this command, it can determine the number of attached devices sharing the same address.

There is of course the possibility that more than 1 device with the same address generates the same random number but the likelihood of this occurring in a small network is low enough to ignore (1 in 254 * 256 = 1 in 65,024).

The random number can be generated by the microcontroller in a number of ways. If it is timer based then there could be a correlation between power-up time and a random number generated at any particular moment in time. Since networked devices could easily share the same power bus, it is preferable to store a pseudo random number in EEPROM during the factory set-up process.

Header 251 - Address change to change the address of the attached device

Transmitted data : [address]
Received data : <none>
This command allows the addressed device to have its address changed for subsequent commands. The host sends 1 data byte, the value of which is the new address. It is a good idea to make sure that 2 devices do not share the same address before sending this command.

Header 250 - Address random to randomly change the address of a device

Transmitted data : <none>
Received data : <none>

This command allows the addressed device to have its address changed to a random value. This is the escape route when you find out that one or more devices share the same address. Randomise their addresses and check them again.

Appendices

A1 The OSI 7-Layer Network Model

The OSI 7-layer network model is of limited use for a simple control protocol such as cctalk. However, for completeness...

Number	Name	Description
7	Application Layer	API & high-level functions
6	Presentation Layer	Transformations (eg. encryption)
5	Session Layer	Network connection open / close
4	Transport Layer	Delivery of information (eg. TCP)
3	Network Layer	Routing & virtual addresses (eg. IP)
2	Data Link Layer	Packet formation (packet switching)
1	Physical Layer	Voltage, pinout, speed, connectivity...

In broad terms...

RS232 deals with layers 1 & 2.

cctalk deals with layers 3 & 4.

There are no session requirements (layer 5) or enforced user encryption (layer 6).

Layer 7 is application specific.

A2 Naming Convention

The following naming convention should be adopted in technical literature to indicate the cctalk interface specification. The idea is to allow some kind of serial comms to be set up given only this number and no other literature (the usual situation in engineering !).

The cctalk id code has 10 features…

cctalk
b **b**aud rate ÷ 100
p interface **p**ort
v supply **v**oltage
a d**a**ta voltage
d supply **d**irection
c **c**onnector type
m **m**aster / slave configuration
x checksum type
i **i**mplementation level
r specification **r**elease

b. The baud rate is either ‘48’ for 4800 baud or ‘96’ for 9600 baud.

p. The interface port is defined as follows :
0 - open-collector interface
1 - *RS485 interface* ← not currently supported

v. The supply voltage refers to the nominal main system power supply voltage to the product, specified as a positive d.c. voltage in volts. There will be an acceptable voltage range associated with each product, for instance 10V to 15V for 12V nominal, together with limits on power supply ripple, noise, spikes, rise-time etc.
5 - 5V
12 - 12V nominal
24 - 24V nominal
48 - 48V nominal

a. The data voltage is the pull-up voltage when using an open-collector interface. cctalk always uses 0V as the active state (start bit condition) but the idle state can be altered to suit the application. The pull-up voltage is always connected via a sizeable resistor (e.g. 10K).
5 - 5V
12 - 12V nominal

It is assumed that for voltages other than +5V obtained via a voltage regulator, the data voltage will track the supply voltage.

d. The supply direction is defined as follows :
0 - supply sink (an external power supply must be connected)
1 - supply source (can be used to power other peripherals)
2 - supply sink or source
c. For connector type, refer to Section 6.

m. The master / slave configuration is defined as follows :
0 - slave device (only replies to cctalk messages)
1 - master device (initiates cctalk messages)
2 - master or slave device (manual switching)
3 - master or slave device (automatic switching)

x. The checksum type is defined as follows :
8 - addition checksum (1 byte)
16 - *CRC-CCITT checksum (2 bytes)* ← not currently supported

i. For implementation level, refer to Section 16.1

r. For specification release, refer to the **major issue number** of this document.

A5.1 MCL Product Example

cctalk Demonstration Board
cctalk b48.p0.v12.a5.d2.c1.m0.x8.i4.r2

Expands as 4800 baud, open-collector, +12V supply, +5V data, supply sink or source, connector type 1, slave device, 8-bit checksum, level 4, release 2

Figure 1 - cctalk Electronics Interface

Version 1

This circuit uses transistor double-buffering to protect the processor from static and noise spikes.

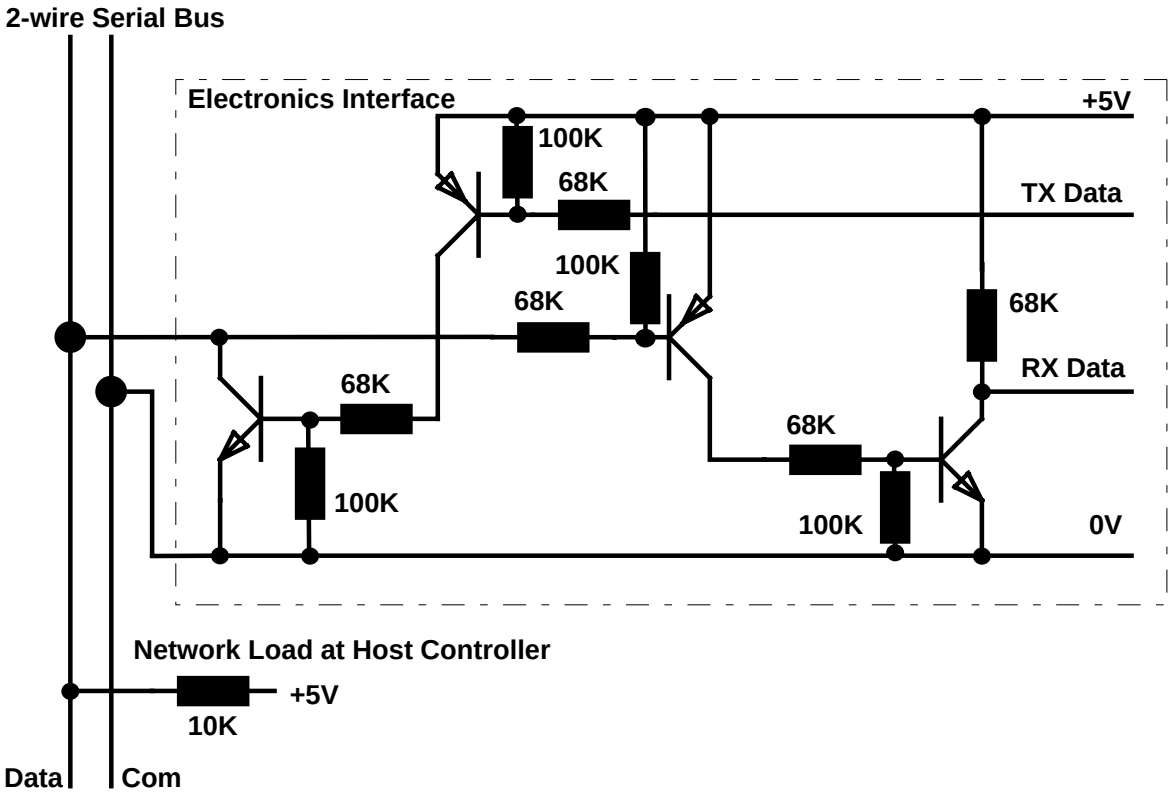


Figure 2 - cctalk Low Cost Electronics Interface

Version 1

Assuming that the transmitting device is capable of sinking a reasonable amount of current, a direct diode interface can be used rather than a full transistor interface. Although cheaper to implement, this circuit does not have the drive capability or the robustness of other designs.

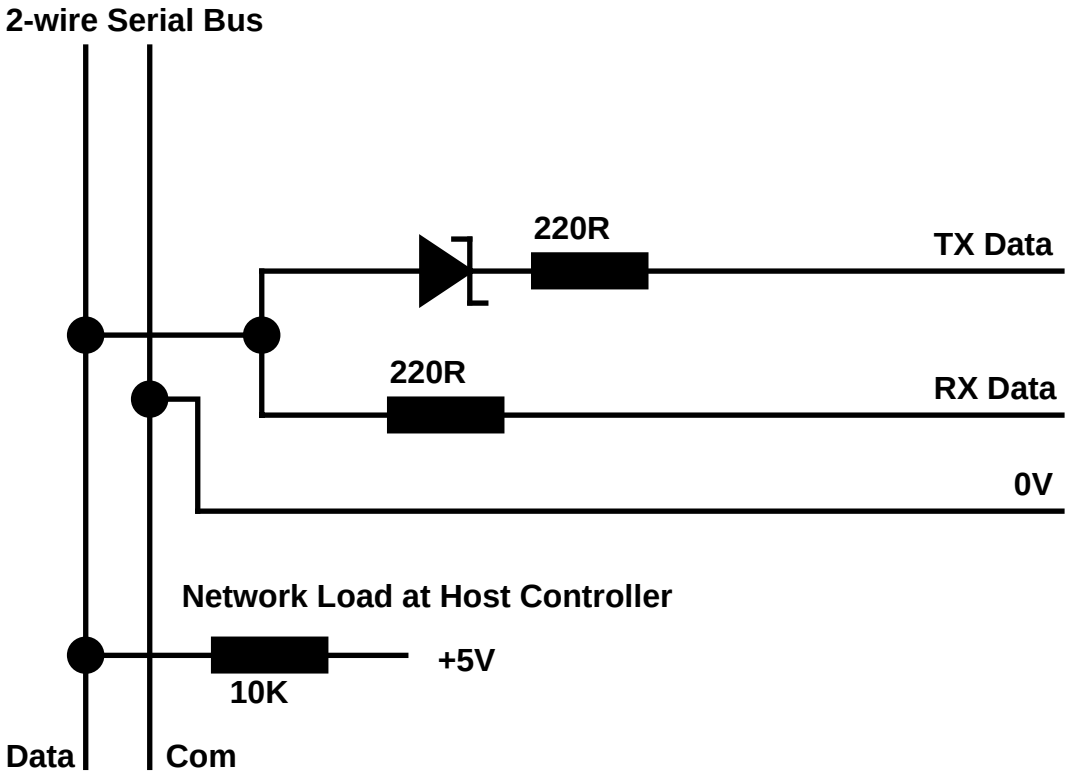


Table 1 - cctalk Command Header Definition Table

Issue 5

The following sheets may be attached to a product specification to indicate which cctalk commands have been implemented.

Product type	
cctalk address	

cctalk id code	cctalk	b	p	v	a	d	c	m	x	i	r	
Header	Supported											Function

255	X	Factory set-up and test
254	X	Simple poll
253	X	Address poll to determine attached devices
252	X	Address clash to determine if any devices have the same address
251	X	Address change to change the address of the attached device
250	X	Address random to randomly change the address of a device
249	X	Request polling priority
248	X	Request status
247	X	Request variable set
246	‘Coin Controls Ltd’	Request manufacturer id
245	‘Coin Acceptor’	Request equipment category id
244	‘CondPlus’	Request product code
242	X	Request serial number
241	‘VPR-V1.00’	Request software revision
240	X	Test solenoids
238	X	Test output lines
237	X	Read input lines
236	X	Read opto states
233	X	Latch output lines
232	X	Perform self-check
231	X	Modify inhibit status
230	X	Request inhibit status
229	X	Read buffered credit or error codes
226	X	Request insertion counter
225	X	Request accept counter
216	X	Request data storage availability
213	X	Request option flags
212	X	Request coin position
210	X	Modify sorter paths
209	X	Request sorter paths
202	X	Teach mode control
201	X	Request teach status
197	X	Calculate ROM checksum
196	X	Request creation date
195	X	Request last modification date
194	X	Request reject counter
193	X	Request fraud counter
192	X	Request build code
185	X	Modify coin id
184	X	Request coin id
183	X	Upload window data
4	1.3.1	Request comms revision
3	X	Clear comms status variables
2	X	Request comms status variables
1	X	Reset device

Table 2 - cctalk Standard Category Strings

Issue 4

The following standard category strings have been defined, along with their typical default addresses. Note that cctalk allows the addresses to be changed to arbitrary values in multi-drop applications.

Evaluation Pod	2
Coin Acceptor	2
Changer	3
Payout	3
Controller	3
Bill Validator	4
Card Reader	5
Display	60
Keypad	70
Power	100

Where code space is limited, the word 'Pod' is an acceptable generic identifier, although it is not very forthcoming as to possible function.

Table 3 - cctalk Error Code Table

Issue 3

The following standard errors have been defined for a coin acceptor and other similar peripherals. Not all may be implemented - please refer to the product manual. A serial credit code indicates that a coin was definitely accepted. A serial error code may or may not indicate a coin was rejected due to most coin acceptors not having a specific reject sensor together with the wide range of possible error trigger conditions (hardware faults, coins getting stuck, deliberate fraud attempts etc.).

Code	Error	Coin rejected ?
0	Null event (no error)	No
1	Reject coin	**Yes - by definition**
2	Inhibited coin	**Yes**
3	Multiple window	**Yes**
4	Wake-up timeout	Possible
5	Validation timeout	Possible
6	Credit sensor timeout	Possible
7	Sorter opto timeout	No
8	2nd close coin error	**Yes - 1 or more**
9	Accept gate not ready	**Yes**
10	Credit sensor not ready	**Yes**
11	Sorter not ready	**Yes**
12	Reject coin not cleared	**Yes**
13	Validation sensor not ready	**Yes**
14	Credit sensor blocked	**Yes**
15	Sorter opto blocked	**Yes**
16	Credit sequence error	No
17	Coin going backwards	No
18	Coin too fast (over credit sensor)	No
19	Coin too slow (over credit sensor)	No
20	C.O.S. mechanism activated (coin-on-string)	No
21	DCE opto timeout	Possible
22	DCE opto not seen	**Yes**
23	Credit sensor reached too early	No
24	Reject coin (repeated sequential trip)	**Yes**
25	Reject slug	**Yes**
128	Inhibited coin (Type 1)	**Yes**
…	Inhibited coin (Type n)	**Yes**
159	Inhibited coin (Type 32)	**Yes**
253	Data block request (note α)	No
254	Coin return mechanism activated (Flight deck open)	No
255	Unspecified alarm code	No

Note α : Special signalling mechanism to support slave requests for data.

Table 4 - cctalk Fault Code Table

Issue 3

Code	Fault	Optional Extra Info
0	OK (no fault detected)	-
1	EEPROM checksum corrupted	-
2	Fault on inductive coils	Coil number
3	Fault on credit sensor	-
4	Fault on piezo sensor	-
5	Fault on reflective sensor	-
6	Fault on diameter sensor	-
8	Fault on sorter exit sensors	Sensor number
13	Coin counting error	-
15	Button error	-
17	Coin auditing error	-
18	Fault on reject sensor	-
19	Fault on coin return mechanism	-
30	ROM checksum mismatch	-
31	Missing slave device	Slave address
32	Internal comms bad	Slave address
33	Supply voltage outside operating limits	-
255	Unspecified fault code	-

Table 5 - cctalk Status Codes

Issue 2

Code	Status
0	OK
1	Coin return mechanism activated (Flight deck open)
2	C.O.S. mechanism activated (coin-on-string)

Table 6 - cctalk Commands by Category

Issue 2

Supervisory & Status
Request equipment category id
Request comms revision
Request manufacturer id
Request product code
Request build code
Request software revision
Request serial number
Request creation date
Request last modification date
Request status
Request polling priority
Request variable set
Request option flags
Reset device
Factory set-up and test

General Diagnostics
Simple poll
Perform self-check
Calculate ROM checksum
Test output line
Latch output line
Test solenoid
Read input lines
Read opto states

Comms Diagnostics
Request comms status variables
Clear comms status variables

Coin Auditing
Request insertion counter
Request accept counter
Request reject counter
Request fraud counter

User Memory
Request data storage availability

Remote Coin Programming
Upload window data

Coin Acceptor
Read buffered credit or error codes
Modify inhibit status
Request inhibit status
Modify sorter paths
Request sorter paths
Teach mode control
Request teach status
Modify coin id
Request coin id
Request coin position

Multi-drop Network
Address poll to determine attached devices
Address clash to determine if any devices have the same address.
Address change to change the address of a device
Address random to randomly change the address of a device