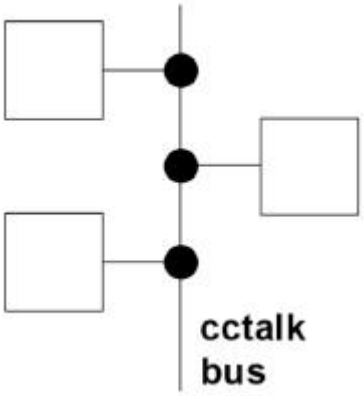


cctalk Serial Communication Protocol
-
Generic Specification
-
Issue 4.4

adherence to, interpretation of, or reliance on, this standard whether it be now or in the future.

Revision History

Issue	Date	Comments
1.0	29-04-96	Draft specification
		< intervening revision history has been archived >
3.1	18-05-99	New section numbering
		Miscellaneous clarifications and additional text
		9600 baud is the preferred operating speed
3.2	03-02-00	Addition of error codes 27 & 28
		Addition of 'Money Controls' to Table 7
		New headers added - see below
		Some headers have new data format for Serial Compact Hopper Mk2
		Revised default addresses : Table 2 - cctalk Standard Category Strings
		Inter-byte delay < 10ms : See 'Timing Requirements'
		Application specific header range is 99 to 20 rather than 99 to 7
4.0	13-06-00	Major document restructuring and text revision
		New headers to support bill validators
	15-05-01	Update to Appendix 10 - Common Country Codes
		Stated conformance to ISO 3166-1
4.1	24-05-01	Modification to recommended cctalk interface circuit
		'Circuit 1 - cctalk Standard Interface'
4.2	05-10-01	Addition of connector type 9 for serial universal hopper
		Serial Protocol - Voltage Levels. Allowable ranges now defined.
4.3	16-04-02	Help text now included for error and fault codes. See Tables 2 & 3.
		ISO 3166 list now fully comprehensive. See Appendix 10.
		Default data voltage is +5V. See Appendix 6.
	02-01-03	Removal of Controller category (= address 3) from Table 1
	05-08-03	Link added to cctalk.org web site
		Change to contact FAX number
		Route code 255 added to cctalk header 154, 'Route bill'
		Clarification of when address is changed with cctalk header 251, 'Address change'.
		Addition of Appendix 11 - Coin Acceptor Messaging Example
		Update of Table 6 - cctalk Standard Manufacturer Strings
	30-09-03	NAK is now a recognised reply from cctalk header 142, 'Finish bill table upgrade' to indicate the process failed.
	30-10-03	Type 8 connector : note added about polarity
	11-11-03	Added Appendix 12 : Italian Flavour Specification Change
		Added latest product naming examples
	19-03-03	Zener versus Schottky diode clarification in Circuits 1 to 4
	29-03-04	Header 163, 'Test hopper'. Added flag explanation.
		Appendix 10 : Common Country Codes : Additional exception
		Addition of header 135, 'Set accept limit'
4.4	06-04-04	Added section on BACTA Token Selection
	05-07-04	Added new bill event code, 'Anti-string mechanism faulty', to Table 7.
		Text '(or reject or other event) ' added to header 162 description.
		Circuit 4 - PC Interface. Alternative transistors given.
	04-08-04	New section : Discussion of Transitory versus Steady-state Events
		Removed from Core Plus : Header 169, Request address mode
		Removed from Core Plus : Header 3, Clear comms status variables
		Removed from Core Plus : Header 2, Request comms status variables
		Calculate ROM checksum : CRC checksum can be calculated with a fixed seed if required
		Request variable set - 2 variables defined for bill validators
		Addition of Appendix 14 - Large Packet Exchange
		Minor text clarification to Header 2, Request comms status variables
		Scope plots added to show cctalk voltage levels and timing
	11-01-05	Text changed to '(strictly speaking an attempted accept sequence)' in

	header 162 description.
25-01-05	Table 1 : Added Reel equipment at address 30
	Table 1 : Added RNG at address 120
	Table 3 : Manufacturer-specific fault code can be sent after ‘255’
	Table 6 : Added Starpoint Electrics to cctalk User Group
	Table 6 : Added Intergrated(sic) Technology Ltd to cctalk User Group
	Added 5 commands for an accumulator hopper, headers 130 to 134
	Expanded status register in header 163, ‘Test hopper’, to include accumulator hopper functionality
	Specification of rise & fall time added
	New connector type 6 part numbers
16-05-05	Re-worked section on BACTA Token Selection
24-08-05	Addition of error code 29 : Accept gate open not closed
	Addition of error code 30 : Accept gate closed not open
	Addition of fault code 42 : Accept gate failed open
	Addition of fault code 43 : Accept gate failed closed
23-12-05	Header 173, ‘Request thermistor reading’ has new Celsius format
	Addition of Header 129, ‘Read barcode data’
	Addition of bill event code 20, ‘Barcode detected’
	Appendix 15 created – Bill Types and Bill Values
28-12-05	Creation of new section ‘cctalk RFC’ in Part 4 / Section 4 of standard.
	Proposed changes to the cctalk specification will be placed here.

Command Headers Added Since Version 4.3

Header 134, Dispense hopper value
Header 133, Request hopper polling value
Header 132, Emergency stop value
Header 131, Request hopper coin value
Header 130, Request indexed hopper dispense count
Header 129, Read barcode data

Command Headers Modified Since Version 4.3

The following commands have been removed from the 'Core Plus' requirement. They are now optional.

Header 169, Request address mode
Header 3, Clear comms status variables
Header 2, Request comms status variables

The status registers in the 'Test hopper' command, header 163, have been increased by one byte for accumulator hoppers and some of the reserved flags re-defined. Operation is fully backwards-compatible with the existing hopper range.

Header 173, Request thermistor reading. Updated return data format.

Part 1 - Contents

1. Historical

Jun 87 : Coin Controls Ltd abandons I^2C development on future products in favour of the RS232 protocol.
Apr 96 : 'cctalk' specification created after much consultation within the industry.
Aug 98 : A meeting of coin mechanism manufacturers in Tamworth, England agrees on a common connector supporting both cctalk and Mars HI^2 for AWP machines. The data format was standardised at +12V, 9600 baud.
Jun 99 : Coin Controls Ltd sets up a cctalk User Group to promote cctalk within the industry and to provide a formal mechanism for both obtaining feedback from users and for expanding the specification into new areas.
Jun 00 : Protocol proving successful in many diverse applications. Specification updated to include an ultra-secure compact hopper and a new range of cctalk bill validators
Aug 00 : Meeting in Burton-on-Trent, England to discuss the future of cctalk in relation to bill validators. Encryption and CRC checksums discussed.
Nov 00 : Encryption and CRC checksums added into the protocol for BNV's. BNV simulation software made available to manufacturers.
Jan 04 : Italy adopts cctalk throughout their AWP platforms and a variety of products are put through homologation. Hoppers are used 'unencrypted'.
Dec 05 : Money Controls successfully tests cctalk running at 1Mbps over a USB virtual COM port link leading to exciting new areas of product development.

2. Introduction

2.1 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 issues when a large number of wires are used to send data. Serial interfaces on the other hand reduce cabling costs to a minimum and often enable extra features (such as self-testing and expansion) to be incorporated into the 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 even 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 aspects 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.

2.2 What is cctalk ?

cctalk (lower-case, pronounced see-see-talk) is the Money Controls (formerly Coin Controls !) serial communication protocol for low speed, control networks. The capitalisation **ccTalk** is the new marketing brand name and should be used on new designs. Note that the following are strongly discouraged :- ccTALK, CCtalk, CCTalk and CCTALK.

The protocol was designed to allow the interconnection of various types of cash handling and coin validation equipment on a simple 3-wire interface (power, data and ground). A basic 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 a RS232-like (the voltages are not RS232 voltages) 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 Money Controls for many years. This means it is a protocol ideal for use in the money 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 or bill sets and the remote FLASH upgrading of firmware.

cctalk can be thought of as achieving the optimal balance between simplicity and security.

2.2.1 Is it a Multi-Master Protocol ?

A multi-master protocol is one which allows more than one device on the bus to be the master, i.e. to initiate a transfer of data. This basically means that any of the slave devices could have a chat with each other. It is a clear intention of cctalk **not to support multi-master mode**. In the money industry today this is seen as the preferred option - the host

machine has total control of the bus and any messaging must be initiated from and conducted by, the host machine. The extra complexity of multi-mastering is not justified and it also creates some security loop-holes with an external data terminal being able to access a peripheral transparently.

There is a discussion of the implications of using cctalk in a multi-master mode in Appendix 5.

2.3 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 and debug 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 necessarily determine the equipment type - it is perfectly possible to have 3 identical coin hoppers 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 - particularly for Microsoft Windows software running on PC's. Control of the 9th bit usually involves non-standard manipulation of the parity bit.

Rather than have a few commands which return large packets of data (an excess of 30 bytes is common in some protocols), 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 Money Controls tells us that customers have widely differing requirements and this approach is best - let them pick and choose from an extensive command list. Managing a large command list is a routine task for most software engineers today.

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. Upper and lower case strings are supported.

Security has always been very important in the protocol. There is now an optional encryption layer and before that there was a mechanism to allow certain commands to be PIN number protected.

2.4 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 a basic command set and limited error handling).

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

- 1K to 3K of ROM
- 30 bytes to 200 bytes of RAM
- 1 x UART - though it can be done in software subject to some timing constraints
- 1 x 16-bit timer

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

Money Controls has written full cctalk modules for the Motorola 68HC05, Hitachi H8 and the Mitsubishi M16C/62 families.

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 9600 baud, each byte transmitted or received takes 1.04ms. 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.

2.5 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. We prefer other manufacturers to refer to **cctalk**Ò in their literature as a recognition of the trademark.

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

possible. If you are interested in registering your interest in cctalk and wish to be kept up to date with the standard then please contact Money Controls.

2.6 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.

2.7 Comparison with other Serial Control Protocols

The table below shows how cctalk compares with some other control protocols…

Protocol	Architecture	Speed	Checksum	Format	Application
cctalk	Bi-directional data line	9600	8-bit *	1,8,1 no parity	AWP
HI^2	Bi-directional data + control	9600	8-bit	1,8,1 no parity	Vending & AWP
MDB	TX + RX	9600	8-bit	1,8,1,1 address bit	Vending
BACTA Dataport	TX + RX	1200 or 9600	8-bit	1,8,1,1 odd parity	AWP
CAN	CANL + CANH	1M	16-bit CRC	-	Automotive
USB	Hubs ! D+ D-	1.5M or 12M	5-bit CRC	-	PC Peripherals
USB2	Hubs ! D+ D-	480M	-	-	Video
Bluetooth	RF wireless	720K	-	-	Consumer

* Optional 16-bit CRC checksum

3. Serial Protocol - Timing

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

9600 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. Error detection is achieved through a packet checksum.

At 9600 baud, each byte takes **1.042 ms**.
At 4800 baud, each byte takes **2.083 ms** - low speed option

3.1 Baud Rate

The baud rate of 9600 was chosen as the best compromise between speed and cost. Higher baud rates require more powerful processors with faster clocks and larger power budgets. 9600 baud is also the most common speed for control network protocols in this industry.

However, there is an option on cctalk to run at 4800 baud on products where the power budget is the overriding consideration. Money Controls has manufactured a low power coin acceptor for a line-powered telephone which operates at 4800 baud.

4. Serial Protocol - Voltage Levels

A level-shifted version of RS232 is used for convenience and to reduce cost. This means negative voltages with respect to the ground rail are not required.

On the serial connector, the idle state = +5V (nominal) and the active state = 0V (nominal).

Mark state (idle)	**+5V nominal**	**Range 3.5V to 5.0V**
Space state (active)	**0V nominal**	**Range 0.0V to 1.0V**

The cctalk interface should see a voltage below 1.0V as an active state and a voltage above 3.5V as an idle state. Voltages in between are indeterminate.

Some older cctalk products had the data line weakly pulled up to +Vs which could be anywhere from +12V or +24V depending on the product power supply. The convention now is to have a +5V pull-up. Which option is used should be clearly documented with the product.

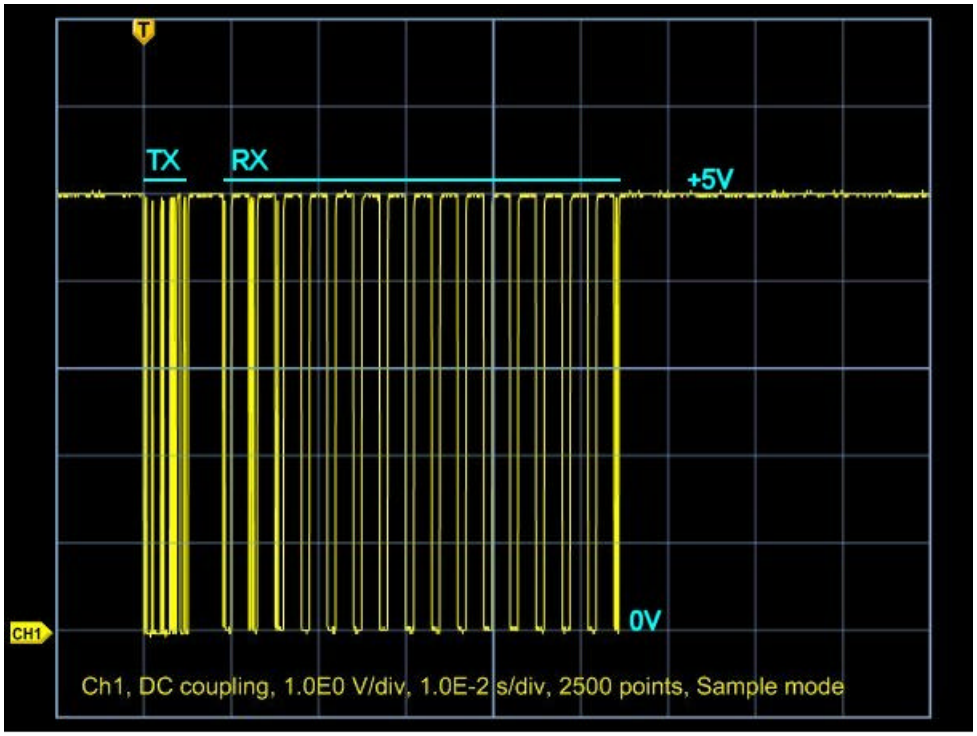
The allowable voltage levels for each state are determined by the interface electronics and these may vary from application to application. The recommended way of driving the cctalk data line is through an open-collector transistor.

The rise and fall time at 9600 baud should be less than **10us**.

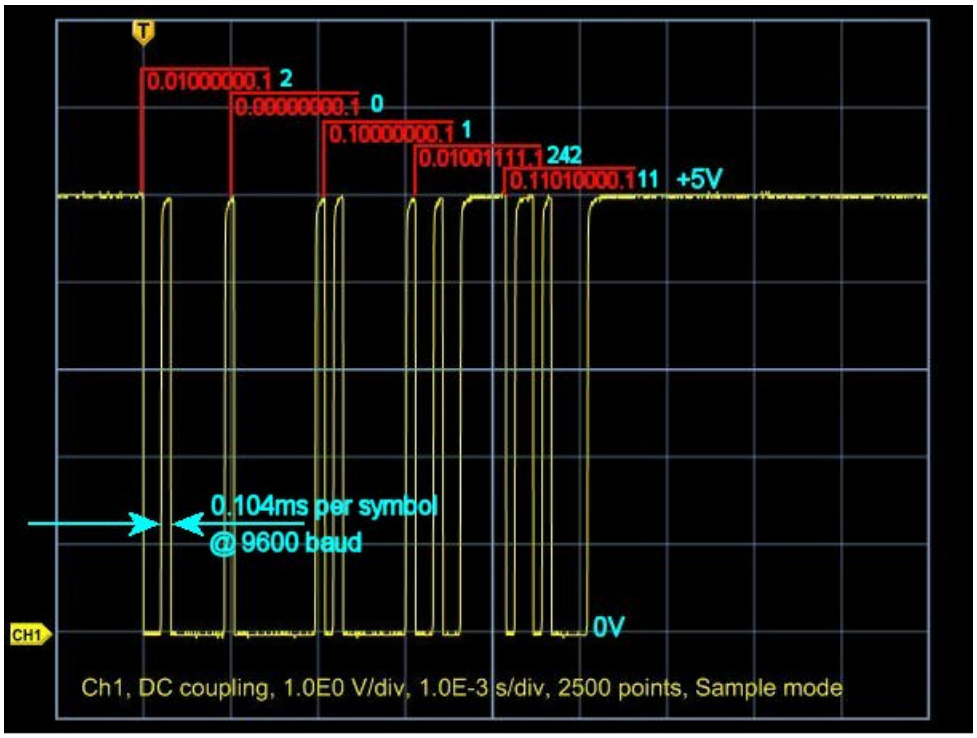
Refer to Circuits 1, 2, 3 & 4.

4.1 Oscilloscope Plots

The top trace shows the 'Read buffered credit or error codes' command sent to a coin acceptor. The TX part is from the host controller to the peripheral and the RX is the reply from the peripheral. The bottom trace shows a single cctalk packet for the 'Request serial number' command with the symbols decoded.



cctalk Data Line - Header 229



cctalk Data Line - Header 242

5. The Serial Data Line

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

The recommended 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 unbalanced 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)	16	1	32
Max. Distance	50ft	50ft	4000ft
Max. Speed	19.2K	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.

The standard cctalk open-collector interface is much simpler to implement than RS485 but less robust when it comes to long distance communication.

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, corrosion resistance, power requirements, cost etc.). It is felt at this stage in the evolution of the cctalk protocol that specifying the connector type is unrealistic and far too restrictive but this situation is under constant review.

The following cctalk connector types have been specified so far…

Type	Pins	Description	Recommended for **new** designs of…
1	4	standard interface, in-line connector	
2	4	standard interface, Molex connector	
3	10	low power interface	
4	6	extended interface, in-line connector	
5	10	AWP industry-standard interface	5 inch Coin Acceptors Bill Validators
6	8	serial hopper interface	
7	4	standard interface, JST connector	3.5 inch Coin Acceptors
8	10	serial hopper interface, version 2	Serial Hoppers
9	12	universal hopper interface	Serial Universal Hoppers

6.1 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)

6.2 Type 2 (standard interface, Molex connector)

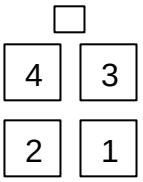
(1) +Vs
(2) <reserved>
(3) 0V
(4) /DATA

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

6.3 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)

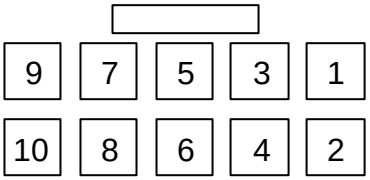
Mechanical keying should be provided by the surrounding cover.

(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 connector from front

6.4 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

6.5 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® from Mars Electronics International) and Money Controls 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 operation
(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.

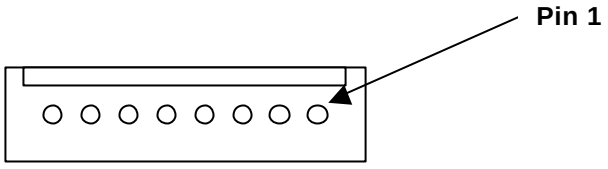
6.6 Type 6 (serial hopper interface)

(1) Address select 3
(2) Address select 2
(3) Address select 1
(4) +Vs
(5) +Vs
(6) 0V
(7) 0V
(8) /DATA

Recommended peripheral connector :
AMP 640456-8 or equivalent e.g. CviLux CI3108-P1V00 or Molex type 6410
(2.54mm pitch)

Mates with :
AMP 640441-8

Pin Polarity :



View of connector from front

6.7 Type 7 (standard interface, JST connector)

(1) +Vs
(2) -
(3) 0V
(4) /DATA

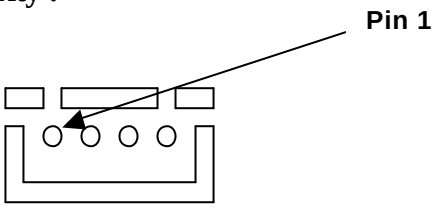
Recommended peripheral connector :

JST B 4B-XH-A

Mates with :
JST XHP-4

Crimps
SXH-001T-P0.6

Pin Polarity :



View of connector from front

6.8 Type 8 (serial hopper interface, version 2)

(1) Address select 3
(2) Address select 2
(3) Address select 1
(4) +Vs
(5) +Vs
(6) 0V
(7) 0V
(8) /DATA
(9) -
(10) /RESET

See connector Type 6 for more details.
AMP 1-640456-0 or equivalent e.g. CviLux CI3110-P1V00.

Note that on the Money Controls Mk2 serial hopper range, pin 1 is on the left with the key at the top, rather than on the right.

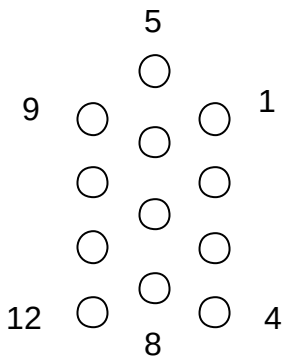
6.9 Type 9 (universal hopper interface)

(1) 0V
(2) -
(3) -
(4) Address select 1
(5) /DATA
(6) -
(7) -
(8) Address select 2

(9) +Vs
(10) -
(11) -
(12) Address select 3

Recommended peripheral connector :
Cinch R76-77848 12-way male

Pin Polarity :



View of connector from front

6.10 Connector Wiring Colours

The following wiring colours have been adopted on some test looms to help debugging.

Signal	Colour
+Vs	Red
0V	Black
/DATA	Yellow
/RESET	Green

7. Message Structure

The protocol now supports CRC checksums and encryption. The differences are shown here.

7.1 Standard Message Packets, Simple checksum

For a payload of 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 the 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 in cctalk conforms to the above structure in the same way all other messages do. Some protocols use a single byte acknowledge - this is not viewed as secure.

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.

For a simple command or request with no data bytes…

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

The acknowledge message is produced by setting the header to zero and having no data bytes…

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

7.2 Standard Message Packet, CRC checksum

[Destination Address]
[No. of Data Bytes]
[CRC-16 LSB]
[Header]
[Data 1]
...
[Data N]
[CRC-16 MSB]

When 16-bit CRC checksums are used, the source address field is replaced by the lower portion of the checksum. In this case, all slave devices automatically reply to address 1 (the default host address).

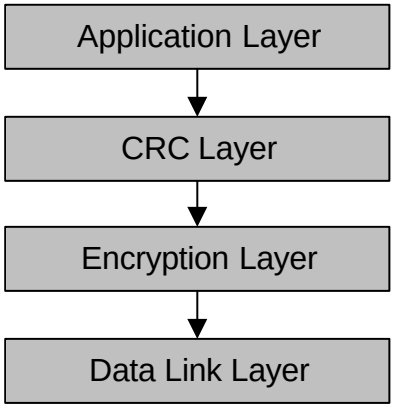
7.3 Encrypted Message Packet, CRC checksum

[Destination Address]
[No. of Data Bytes]
[Encrypted 1]
…
[Encrypted N]

When encryption is used on top of the CRC checksum, all bytes are encrypted from the first checksum byte onwards. The only unaffected bytes are the destination address and length byte which need to remain as they are to allow standard and secure peripherals to be mixed on the same bus. An encrypted message will be ignored by all peripherals with no address match.

7.4 Protocol Layering

In terms of how the different protocol layers are implemented in cctalk…



To send a message to a bill validator, we take the high level command consisting of a destination address, length field, command header and data bytes. A 16-bit CRC checksum is calculated and added into the structure. This is then encrypted and passed to the UART which adds in start and stop bits for transmission. The message is then sent to the peripheral device which performs the operation in reverse.

7.5 Destination Address

Range 0 to 255 (254 slave addresses)

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

7.5.1 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' (when simple checksums are used) 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 in the host software by ignoring bus activity for a fixed time after the command is sent. The broadcast address is best reserved for a couple of the MDCES commands.

The reason that data is returned on a broadcast address is one of huge convenience when debugging software. A typical slave device may have a number of address selection methods, some of which write values into EEPROM, and it may not be obvious which one is active at any one time. Assuming only this device is connected to the serial bus then the broadcast message can be used to guarantee an address match.

7.6 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.

7.6.1 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. A nice power of 2 is usually preferred by software engineers.

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

A good example of block transfer is the 'Read data block' command and the 'Write data block' command. Also, see **Appendix 14**.

7.7 Source Address

Range 1 to 255.

The default source address for the host machine is 1 and there should be no reason to use a different value.

When a slave device replies to the host, the source address is set to that of the slave.

To clarify with an example…

Message from host to slave
Destination = 3, Source = 1

Reply from slave to host
Destination = 1, Source = 3

If CRC checksums are used, there is no source address…

Message from host to slave
Destination = 3

Reply from slave to host
Destination = 1 (assumed to be the case)

7.8 Header

Range 0 to 255.

Header bytes have been defined to cover a broad range of activities in the money industry.

There is now comprehensive support for…

- Coin Acceptors
- Bill Validators
- Serial Hoppers

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.

7.9 Data

Range 0 to 255.

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

Refer to each specific command for its associated data format.

7.10 Simple 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.

If a slave device receives a message intended for another device (destination address does not match) then it should ignore the checksum.

7.11 CRC Checksum

The CRC checksum used in cctalk is the 16-bit CRC-CCITT checksum based on a polynomial of x^16+x^12+x^5+1 and an initial crc register of 0x0000.

Refer to Appendix 9 for more details.

7.11.1 A Little 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.

7.12 Anatomy of an Example Message Sequence

Let's suppose the host machine wishes to find the serial number of an attached slave device. It therefore sends the 'Request serial number' command.

Host sends…
[2] - this is to slave address 2
[0] - there are no additional data bytes to send
[1] - this is from host address 1
[242] - header is 242 (Request serial number)
[11] - checksum, 2 + 0 + 1 + 242 + 11 = 256 = 0

Host receives…
[1] - this is to host address 1
[3] - there are 3 data bytes in the reply
[2] - this is from slave address 2
[0] - header is 0, i.e. it's a reply !
[78] - data byte 1 = 78
[97] - data byte 2 = 97
[188] - data byte 3 = 188
[143] - checksum, 1 + 3 + 2 + 0 + 78 + 97 + 188 + 143 = 512 = 0

Interpreting the return data,
serial number = 78 + 256 * 97 + 65536 * 188 = 12,345,678 in decimal.

8. 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.

An ACK = [Destination Address]
[0]
[Source Address]
[0]
[Checksum]

8.1 The NAK Message

The NAK message has restricted 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**.

8.2 The BUSY Message

The busy message can be used by a slave device to indicate that it is busy and can not respond to, for example, a request for information. It is then up to the host software to retry after a suitable interval. This response is provided for completeness only and is **not currently used** in any Money Controls products.

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

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

9. 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

The maximum string length is 252 characters in line with the maximum number of data bytes in a packet.

9.1 Fixed length Strings

Some commands return fixed length strings for convenience. Where the stored string is less than the fixed string width it is usual to right pad the string with either…

a) spaces (ASCII 32)

b) dots (ASCII 46)

10. 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 electronics. The software for a UART implementation will have to deal with transmitted data being immediately received back again (local 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 main application code.

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 >

11. Timing Requirements

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

11.1 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. Serial 'garbage' must not be left hanging around in the receive buffer.

When transmitting a cctalk data packet, serial communication software should ensure the smallest possible inter-byte delay. At 9600 baud, each byte takes 1.04ms and so the inter-byte delay should ideally be **less than 2ms** to maximise the bus message bandwidth, and certainly **no greater than 10ms**. This is particularly important in a multi-drop application where a number of different peripherals have to be serviced within a set time. A slow slave device will compromise the integrity of the entire bus.

11.2 Between command and reply

The delay between the issuing of a command and the reply being received is command 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.

12. 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.**

12.1 Retransmission

Some protocols have retransmission capability built into the transport layer (see Appendix 2). 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 more 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.

13. 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.

For example, if a coin acceptor is asked to pay out coins with the ‘Dispense hopper coins’ command, then this is clearly an impossible task and there will be no reply.

14. Practical Limitations in Very 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 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.

15. Command Set

15.1 Command Expansion

One major feature of cctalk is limitless command expansion and this part of the protocol has received a lot of attention. There are two methods by which this can be achieved.

15.1.1 Expansion Headers

Headers 100, 101, 102 and 103 are used to indicate another set of headers within the message data. Although this lengthens the expansion messages by 1 byte, it immediately gives access to 1024 extra commands. Money Controls will use this approach for its future range of products.

As an example, suppose we add a new command 'Request ASCII serial number' which returns a serial number in ASCII rather than binary. We will define this new command as EH100:255 (expansion header 100, sub-header 255).

Host sends…
[2] - destination address
[1] - 1 data byte = 1 x sub-header
[1] - source address
[100] - expansion header 100
[255] - sub-header 255 (e.g. Request ASCII serial number)
[153] - checksum, 2 + 1 + 1 + 100 + 255 + 153 = 512 = 0

Host receives…
[1] - destination address
[8] - 8 data bytes
[2] - source address
[0] - reply header
[49] - '1'
[50] - '2'
[51] - '3'
[52] - '4'
[53] - '5'
[54] - '6'
[55] - '7'
[56] - '8'
[81] - checksum, 1 + 8 + 2 +… = 512 = 0

15.1.2 Context Switching

Headers 99 down to 20 are application specific. The way this works is that, for example, header 99 performs a number of different tasks depending on the type of peripheral it is. So the host machine must identify the peripheral type first (is it a coin acceptor, bill validator,

payout…?) and then use the appropriate header number. In this way, we can have 80 commands on each new type of device.

Context switching is the recommended way for other manufacturers to introduce additional features outside the official command set.

Note that command header 255 is the 'Factory set-up and test' command. This command can be used by any manufacturer to implement a range of proprietary functions.

15.2 Implementation Level

The cctalk implementation level is a number (1 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 such that the host machine can use both old and new versions of the product. It is a design aim 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 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, refer to the 'Request comms revision' command. The first byte returned is the cctalk level.

16. 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. Extensions are now in place for payouts and bill validators. 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 along 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. Special header numbers are reserved for this purpose.

17. Implementation Standards

With serial communication being a relatively new feature of the AWP industry, standards are still in the process of being firmed up. One of the key areas for agreement amongst the machine manufacturers must be the choice of connector and a common command set.

For details of which connector to use, refer to the 'Connector Details' section. The table shown represents the state of play.

For which commands should be implemented on each peripheral type, refer to the master cross-reference chart in Appendix 1.

For recommended default addresses refer to Table 1.

18. Coin Acceptors - Credit Polling Algorithm

In a typical coin handling application with a single coin acceptor, only 1 command needs to be sent regularly by the host machine. This is the 'Read buffered credit or error codes' command.

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

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)

Each poll returns between 0 and 5 new credits or other events.

18.1 The Credit Poll Watchdog

A new feature of cctalk serial devices is the addition of a 'credit poll watchdog'. Credit polling involves sending the 'Read buffered credit or error codes' command to coin acceptors or the 'Read buffered bill events' command to bill validators. If for some reason the host communication link dies, it is essential that coins or bills are not swallowed (physically accepted but no credit given). This can be prevented by the peripheral having a watchdog timer triggered off the serial poll command. If the polling stops, the peripheral goes into an inhibit state. Once polling resumes, the inhibit state is lifted.

19. 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 product code	ASCII	What model is it ?
Request build code	ASCII	What build is it ?
Request manufacturer id	ASCII	Who manufactures 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.

19.1 Designing a cctalk API

It should be a straight-forward task in most languages to write a cctalk library which hides the messy process of actually sending and receiving bytes through the UART. Once this is written, a simple API can make the task of adding extra cctalk commands a painless operation.

A simple example is shown below. We define global variables or public members which hold the destination and source addresses for host communication. This saves having to pass them into the send function each time. To fire off a cctalk command, we pass the command header number and an array of parameters which vary according to the command. The function can return a boolean to indicate a successful outcome or if there wasn't, a comms error in the status string e.g. 'Error : Bad checksum'. Some messages will return an ACK (no receive data) and others will return an array of command-specific data.

```
Global Variables
destAddress = 2
srcAddress = 1

Functions
BOOL SendcctalkCommand( commandHeader, noParamBytes, paramBuffer[],
noRxDataBytes, rxDataBuffer[], commsStatus )

If TRUE then
      ACK or rxDataBuffer contains returned data

If FALSE then
      Error string in commsStatus
```

The entire cctalk command set is covered by this simple interface (apart from a couple of MDCES commands) and a single line of source code executes the complete message transfer.

Extra parameters can be added to allow the serial port to be selected, the baud rate to be changed and the receive timeout period to be varied according to the command. There may also be options to change the checksum method and to enable and disable encryption.

20. Multi-Drop Considerations

The cctalk 3-wire serial interface has been designed to allow a number of peripherals to be joined together with the minimum of effort. There is no restriction on connection topology - they can be linked in-line, in a ring or in a tree structure. This environment is typically referred to as 'multi-drop' since each device is dropped off the bus.

A multi-drop environment is obviously a lot more complicated than a simple interconnection between one master device and one slave device. For a single master, multiple slave configuration, the major problem is address resolution. All slave devices must have a unique address on the bus for the system to work properly.

The most obvious first step is to ensure that different peripheral devices have different default addresses. Looking at Table 1 it can be seen that coin acceptors default to address 2, hoppers to address 3, bill validators to 40 etc. A problem arises when more than one peripheral is added of a particular type. For instance, it would not be unusual in a vending machine to have 3 hoppers paying out change. As it would not be practical for peripheral manufacturers to produce a range of build variants having different default addresses, together with the problem of the units getting mixed up when they look identical, it is preferable on these devices to have a means of external address selection. This might be a DIP switch on the PCB or address selection via the wiring harness on the connector. The wiring harness method is particularly convenient for machine manufacturers as it means that a physical position within the machine will always have the same address.

To support more advanced applications such as batch programming of coin acceptors, cctalk has command support for dynamic addressing. This basically means that device addresses can be resolved and changed via software means alone (assuming the device supports dynamic addressing). So we could plug 10 identical coin acceptors all with address 2 into a PC and test each one individually.

To provide comprehensive multi-drop support we have the MDCES commands - see the section below. Many multi-drop networks are determinate (addresses known) and so these extra commands are not needed.

20.1 Practical Limitations of Multi-Drop Networks

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 or threshold voltage comparitors.

The maximum length of connection will depend on the degree of electrical noise (radiated and conducted) in the system. We are operating at relatively 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.

20.1.1 Maximum Number of Network Devices

The following restrictions apply when connecting lots of peripherals to the cctalk bus.

20.1.1.1 Logical Addressing

The protocol allows the logical connection of **254 devices**.

20.1.1.2 Electrical Loading

Each device added to the bus places an electrical load on it. The extent of the loading will be application dependent but taking a typical bus pull-up resistor of 10K to +5V, if each device receiver sinks 10uA then a total of **20 devices** would lower the bus voltage by 2V. Any more drop than this would seriously compromise the noise immunity.

20.1.1.3 Address Randomisation

This section only applies if devices are not initialised with unique addresses…

For the MDCES randomise command to have a good chance of **not** setting 2 devices to the same address, it is suggested no more than **8 devices** are connected to the bus. The statistical chance of a clash occurring is about 1 in 10. In which case, the network would have to be randomised again which delays the start-up sequence by a couple of seconds. The chance of more than a 5s delay would be about 1 in 1000.

Refer to Appendix 8 if you are interested.

20.2 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 most situations it is possible to have device addresses configured before use so that no address ambiguities arise. In a few situations, new devices will be plugged into the network without prior knowledge. The host controller needs then to 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 and the 'Address clash' to determine if any devices have the same address. 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 very easily. Obviously there is no problem for a pre-configured network with all address ambiguities resolved beforehand.

Headers 253 and 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).

20.2.1 Address Poll, Header 253

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

The algorithm produces an overall delay of 1200ms during which time the receive port is ignored to avoid picking up a cctalk message packet composed of seemingly random bytes.

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

20.2.2 Address Clash, Header 252

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) // range 0 to 255
Disable rx port
Delay (4 * r) ms
Send [addr]
Delay 1200 - (4 * r) ms
Enable rx port

The algorithm produces an overall delay of 1200ms during which time the receive port is ignored to avoid picking up a cctalk message packet composed of seemingly random bytes.

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). An address clash is easy to pick up later as comms errors would be plentiful.

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 may be preferable to store a pseudo random number in EEPROM during the factory set-up process.

20.2.3 Address Change, Header 251

Transmitted data : [address]
Received data : ACK

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. A full ACK message is returned.

Note the ACK is sent back from the original address, not the changed address. In other words, the change to the cctalk address field is done **after** the ACK is returned rather than before.

e.g.

```
TX = 002 001 001 251 003 254 - Change from address 2 to
3
RX = 001 000 002 000 253 - ACK from address 2
```

20.2.4 Address Random, Header 250

Transmitted data : <none>
Received data : ACK

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. A full ACK message is returned.

To simplify host software, any address clash is best dealt with by randomising the entire network with the broadcast address. In this case, all peripherals will reply together with an ACK which will be seen as garbage due to clashing. The host software should therefore ignore all return bytes and communication errors immediately after a broadcast randomise command.

Slave devices should not allow the random address value to be changed to 0 (the broadcast address) or 1 (the host address).

Note the ACK is sent back from the original address, not the changed address. In other words, the change to the cctalk address field is done **after** the ACK is returned rather than before.

21. Discussion of Transitory versus Steady-state Events

The primary mechanism used to transfer information from a peripheral to the host machine is through event polling.

For coin acceptors this is…
Header 229, Read buffered credit or error codes

For bill validators this is…
Header 159, Read buffered bill events

For hoppers this is…
Header 166, Request hopper status

The hopper is slightly different in that the status information just contains details of coins paid and coins unpaid. The coin acceptor and bill validators can both report events as well as credits.

Events can either be transitory such as…
- Coin going backwards
- Credit sensor timeout
- Invalid bill (due to validation fail)
- Stacker inserted

But there are other events which can possibly be permanent…
- Credit sensor blocked
- Sorter opto blocked
- Bill jammed in transport (unsafe mode)
- String fraud detected

The philosophy of cctalk is to only report 'set' events back to the host but not 'clear' events. This cuts down on the number of events flying around - 'String detected' is not followed by 'String not detected' etc. Events are not bracketed but are single-shot notifications of anything unusual. To find out whether an event is still occurring, the host should switch to a steady-state status command in an alternative polling loop. The steady-state status is provided by the diagnostic commands available on all peripherals.

For coin acceptors & bill validators these are…
Header 232, Perform self-check

For hoppers this is…
Header 163, Test hopper

The hopper test can be done if the paid coins (when the motor stops) is less than the value requested. The reason for the underpay can be discovered by looking at the hopper status flags. This may be due to the hopper bowl being empty or a deliberate fraud attempt.

So how does this actually work in practice ? See below.

Device-side Operation

```
Fault ‘A’ develops
Self-inhibit to prevent further acceptance of coins or notes
Latch fault code ‘B’
Report event ‘A’ when polled ( single occurrence )
Report fault ‘B’ when polled
```

If the fault goes away then…

```
Re-enable to resume acceptance of coins or notes
Report fault ‘OK’ when polled
Report new events when polled
```

Host-side Operation

```
Poll events returns event ‘A’
Change polling loop to self-check diagnostics
Fault poll returns ‘B’
```

If the fault goes away then…

```
Fault poll returns ‘OK’
Change polling loop to events
Poll events returns new events as they occur
```

Operation is ‘safe’ in that the host machine is not required to take any action in the event of a fault as the peripheral device will self-inhibit immediately. The ‘Modify inhibit status’ and ‘Modify master inhibit status’ commands do not have to be sent.

The diagnostics loop should be reasonably tight (poll at least once per second) in case the device recovers and new credit events are waiting to be polled. If the host wishes to permanently shut down the device on first indication of a fault it can always issue a master inhibit.

As can be seen, event code ‘A’ is mapped into a fault code ‘B’. Looking at the list of event codes (Table 2, 7) and fault codes (Table 3) it can be seen that a one to one mapping does not always exist. Fault codes generally relate to a physical component or sensor whereas event codes relate to an activity or assumed activity. But it should be straight forward to map one into the other and system integrity will not be compromised by an incorrect mapping. It is merely data logging activities that are affected.

For instance, we could map an event code of ‘Credit sensor blocked’ to a fault code of ‘Fault on credit sensor’. Whether the fault is due to a coin blocking the credit sensor or a fault with the credit sensor itself does not really matter as far as the logic is concerned. In fact, it is unlikely the coin acceptor itself can distinguish between the two conditions.

In actual operation, we could have a transitory ‘Credit sensor blocked’ event due to a slow or partially trapped coin. If the condition remains then the coin acceptor will report a ‘Fault on credit sensor’ until the coin frees itself or a service engineer is called out to investigate further.

In this way, both transitory and steady-state events are handled by cctalk.