Last updated 21/08/02

ccTalk Frequently Asked Questions

Money Controls

Your questions have been answered by the ccTalk® system architect, Andy Barson.

1. Diary of changes

Issue 1.0……..……..Aug 2002

- 1st Issue

2. General Questions

2.1 Where does the name 'ccTalk' come from ?

The protocol was developed at Coin Controls before the company changed its name to Money Controls. Hence Coin-Controls-Talk.

2.2 Which is correct - 'cctalk' or 'ccTalk' ?

The name 'ccTalk' is the brand name and should be used by all manufacturers on labels and in technical manuals. The original specification was created by the author as lower case 'cctalk' to contrast with all the upper case protocols around at the time. Most of the specification documentation refers to 'cctalk'.

2.3 Does the name ccTalk refer to hardware, software or a written specification ?

All three. In essence the protocol as laid down by the written specification.

2.4 What is ccTalk and who uses it ?

ccTalk is a serial protocol for use in the Money Processing Industry. It can be viewed as a low speed, control network with a wealth of features covering all aspects of secure credit transfer, status reporting, fault reporting and host control across a wide range of industry-standard peripherals. It is used throughout the world in sectors such as amusement, video, gaming, transportation, vending, telecommunications and retail.

2.5 Where do I get information on ccTalk ?

The most important document is the 'ccTalk Serial Communication Protocol - Generic Specification' which explains how the protocol works and the history behind it. More detailed information can be found in product manuals for each ccTalk peripheral. The latest copy of the generic specification can be obtained directly from Money Controls or via the ccTalk web site at *www.cctalk.org*.

2.6 Who is the controlling authority for ccTalk ?

It is currently with Money Controls who originated the specification but this is subject to review. Please send any comments you have on the specification or suggestions for improvements or extensions to *abarson@moneycontrols.com*

2.7 Do I have to pay any fees to use it ?

No. The ccTalk protocol is 'gifted' to the industry and does not require a license fee or royalties to be paid. There are no restrictions on use.

2.8 How many wires are needed on ccTalk ?

A standard ccTalk serial bus only requires 3 wires. There is a supply voltage line, a common 0V line and a bi-directional data line. Some applications require additional power supply voltages and / or control signals and address lines.

2.9 What peripherals are covered by ccTalk ?

The specification covers coin acceptors, bill validators and payout devices such as hoppers. There are plans to expand the command set to card payments and ticket printers.

2.10 Why are the ccTalk command headers not in a sensible order ?

Hindsight is a wonderful thing ! The ccTalk headers were created from 255 downwards and as needs arose. They are roughly in the order coin acceptors, hoppers and bill validators but with some turbulence. Ask any software engineer and she doesn't care - they are typed into an include file or global module and then forgotten about. We tend to remember names much better than numbers.

2.11 What software support can you provide ?

Money Controls is primarily a hardware manufacturer and does not provide a suite of tools for software development. We can provide specification documents and coding examples but we do not supply software libraries, DLL's, OCX's, API's etc. It is up to each manufacturer to write their own ccTalk software. Is ccTalk the same as RS232 ?

This is a 'yes' and 'no' answer. The underlying protocol is the same as RS232 in terms of the asynchronous transfer of characters with start and stop bit timing frames. However, two changes have been made to reduce the cost of implementation on embedded peripherals. The transmit and receive lines on RS232 have been combined into a single bi-directional data line and the mark / space voltages of ±12V have been changed to 0V / +5V.

2.12 I've heard about ccTalk encryption. What is it and when is it used ?

The original ccTalk protocol did not use any kind of encryption. It was thought that the security in a serial interface was so much better than a parallel one that no further steps were necessary. However, a perceived threat was seen in the ability to empty a full hopper bowl of coins with a simple connection to the multi-drop bus, as well as the ability to clone high value credit packets on a bill validator. Therefore various levels of encryption were added seamlessly to the ccTalk protocol. Hoppers use a security key to unlock the dispense command and bill validators only operate in secure ccTalk mode with encryption on every command.

2.13 Can ccTalk run at anything other than 9600 baud ?

The standard ccTalk baud rate is 9600 and it is recommended that all peripheral devices use this speed. The cost of supporting higher baud rates on small embedded devices can be prohibitive. The operation of ccTalk at the packet formatting level is unaffected by the baud rate and so the generic specification indicates that 4800 and 19,200 in certain applications would be an option. The mixing of baud rates on a multi-drop bus is not acceptable however as it would result in too many errors.

2.14 Has anybody written a ccTalk protocol analyser ?

Money Controls has a basic software package. It is a simple comms monitoring application which runs on a PC under Windows and decodes all data appearing on the RX pin of the RS232 connector according to ccTalk rules. It is available on request but supplied 'as is' - without instructions or warranty.

2.15 How many peripherals can be supported on a ccTalk bus ?

This is an 'it depends' type answer. The address field of ccTalk is 1 byte in size. The value of 0 is reserved as a 'broadcast' address. The value of 1 is the default source address of the host machine. So that leaves a theoretical possibility of connecting 254 slave devices. In practice we cannot get anywhere near this figure for two reasons - bandwidth and electrical loading. If all the peripherals require polling every 1 second then the time slot available for each peripheral would be 3.9ms. This is not long enough for a typical ccTalk message to complete. If the peripherals only require polling when a certain action is being performed (e.g. dispensing coins from a hopper) then this limitation could potentially be removed. Electrical loading is the result of each ccTalk peripheral lowering the impedance of the ccTalk data line. The extent to which this is done depends on the exact interface electronics used, but with a typical configuration it is possible to connect 10 to 20 devices.

2.16 What is the expected ccTalk data line voltage ?

The idle state of the ccTalk data line is nominally +5V but will be slightly less than this due to electrical loading. Anything between 4V and 5V should be treated as a 'high' by the interface electronics. Anything below 1V should be treated as a 'low'. Some early ccTalk products from Money Controls had the data line voltage pulled up to the supply voltage of +12V or +24V but this is now discouraged.

2.17 How 'noise-immune' is ccTalk ?

How long is a piece of string ? The question can be looked at in terms of electronics and software. On the electronics side, the ccTalk data line is driven by an open-collector transistor onto a low-voltage wire with a weak pull-up resistor. So compared to RS485 which uses differential current drivers and a balanced line, we have a susceptible system. In terms of software however, there are safeguards such as CRC checksums and infinite retries which allow any burst of electrical noise to only temporarily disrupt serial communications. In that sense ccTalk is a 'resilient' protocol, even if response times cannot be guaranteed in noisy environments. However, the criticality of any single-shot process such as a bill credit event has been removed in the upper layers of the protocol through event buffering.

In short it is recommended that ccTalk is used for 'in-machine' hook-up of peripheral devices where the total length of the ccTalk data wire is less than 10m. For connection between machines and between sites another physical layer should be used such as RS485, Ethernet, modem etc. In principal there is no reason why ccTalk cannot be run over these other layers without change as there are very few timing requirements above those of the command and response loop delay.

2.18 Can I run ccTalk over USB ?

The simple answer is yes. If you know about the different 'classes' that are available on USB then you may have heard about the CDC or 'COM class'. This refers to the software required to convert a high-speed data link provided by the USB hardware into a RS232 emulation. So using a COM class converter, ccTalk can be run over USB 'transparently' - as if the USB pipe was not there. The speed advantages of USB would not be apparent however as there would be a 9600 baud bottleneck on existing peripherals.

2.19 Is there a ccTalk test house or approval facility ?

Not at this point in time.

2.20 When was ccTalk 'invented' ?

We prefer to say that ccTalk 'evolved' from earlier protocols and after much consultation within the industry, rather than magically appear on a particular date. The earliest ccTalk labelled specification was generated in 1996.

2.21 Is there a web site for ccTalk ?

Yes - visit *www.cctalk.org*

2.22 What kind of timing problems can I get with ccTalk ?

The timing of ccTalk essentially boils down to firing a command packet to the peripheral and waiting for a reply to come back. Within each packet there is an expected maximum delay between bytes. So subject to these two 'timeout' conditions, no other timing problems should arise, unless specifically documented with the peripheral.

A typical transfer would see the host machine send a message to the peripheral. If no response is obtained within 1 second (this could be much shorter depending on the command) the host could try again. When the reply is being received, any gap of more than 50ms between received bytes would force a reset of the receive pointer. In other words, it would cause the current message packet to be abandoned and a new one started with the 'destination address' assumed to be next.

Commands on a multi-drop bus can be sent 'nose to tail'. In other words as soon as the host receives the complete return packet from peripheral A, it can send the next command to peripheral B with zero delay between. So the start bit of the peripheral A destination address can come immediately after the peripheral B checksum stop bit. In practice there will probably be a delay of a few milliseconds.

2.23 My ccTalk software doesn't work. What should I do ?

The reasons could be many and varied so it is best to start with the simplest possible ccTalk command and work up from there. The ccTalk header 254 is a 'Simple poll'. The peripheral replies with an ACK and all ccTalk peripherals must support it.

For a peripheral on address 2 and assuming 8-bit checksum and no encryption…

Host sends [2] [0] [1] [254] [255]
Slave returns [1] [0] [2] [0] [253]

The values between brackets are bytes with the decimal value shown.

If there is no response from the slave then check the following…

- Is there power on the ccTalk +Vs line ?
- Is the peripheral operating in serial mode ? There could be a DIL switch or connector pin option.
- Is the ccTalk data line high in idle ?
- Does the ccTalk data line go low during message transmit ?
- Is each low bit about 1ms (9600 baud value) ?
- Are the signal transitions fast and clean ? Check with an oscilloscope.
- Does the peripheral use CRC checksums and encryption ? If so the above example will not work !

2.24 How do I go about obtaining ccTalk encryption documents ?

The encryption documents are sensitive and will not be made available in the public domain. If you require the documents for serial hoppers or bill validators then contact Money Controls and we will send out the necessary paperwork for signing prior to you being sent a copy by post. The use of email to circulate these documents is strictly prohibited.

2.25 How does ccTalk perform error correction and retries ?

There is nothing in the transport layer of ccTalk to assist in the automatic retry of messages with bad checksums - this has been left entirely to the discretion of the application layer. This has advantages in that the host software can be made as simple or as complicated as the application requires. The host can retry once, 3 times, 10 times, for 1 second, for 10 seconds etc. It can even have infinite retry.

The conditions for message retry are based on the following conditions…

- No response is received from the slave (the standard ccTalk error condition)
- A NAK message is received from the slave
- There is a low-level RS232 framing error (stop bit invalid)
- There is a data underrun or overrun based on the length field
- The message is received with a bad checksum
- The message is received with an incorrect address or header field

The ccTalk commands are structured such that infinite retry can be attempted without penalty. Rather than the ‘toggle bits’ that some protocols use to distinguish between an original message and a retry message, ccTalk uses a full-byte event counter to prevent single-shot events from being re-issued or mis-counted. Only a few commands are subject to the software overhead of an event counter - the rest do not need it and do not have it.

2.26 My 8-bit checksum is wrong. Why ?

If you send a message to a ccTalk peripheral with an incorrect checksum then it will not reply.

The 8-bit checksum is calculated by adding up all the message bytes from the destination address to the last data byte and finding the value when added to it will produce zero in modulo 256 arithmetic.

For example, the ccTalk command to enable all coin inhibits is…

[2] [2] [1] [231] [255] [255] [checksum]

2 + 2 + 1 + 231 + 255 + 255 = 746 = 234 modulo 256 (256 x 2 + 234).

Checksum = 256 - 234 = 22

Therefore the complete message is

[2] [2] [1] [231] [255] [255] [22]

2 + 2 + 1 + 231 + 255 + 255 + 22 = 768 = ZERO modulo 256.

2.27 My CRC checksum is wrong. Why ?

If you send a message to a ccTalk peripheral with an incorrect checksum then it will not reply.

A 16-bit CRC may used in ccTalk, positioned in the message packet as follows…

[Dest. Addr.] [Size] [CRC-16 LSB] [Header] [Data 1]… [Data N] [CRC-16 MSB]

It can be seen that the 'Source Addr.' field of ccTalk has been replaced by the lower half of the 16-bit checksum. This is possible because the source address is redundant in a single-master system. It is advantageous to keep message lengths the same so that CRC and non-CRC protocols can be mixed on the same bus.

The CRC algorithm has a number of options and you need to have them exactly right for it to work in all cases.

The ccTalk protocol uses the following parameters…

- CRC-CCITT
- Polynomial = x^16 + x^12 + x^5 + 1
- Initial crc register = 0x0000

The algorithm in 'C' is included in the appendix of the generic specification. It is necessary to have some programming experience as calculating it by hand is no fun.

A simple poll would have the following CRC data…

TX : [40] [0] [a] [254] [b]

a = 182 b = 33

RX : [1] [0] [a] [0] [b]

a = 48 b = 55

2.28 What is the longest ccTalk message size ?

The 2nd byte of a ccTalk message is the 'no. of data bytes'. This is not the total size of the message packet but the size of the data payload only. A message with no data bytes would still be 5 bytes in size. It would be possible to have 255 data bytes such that the maximum packet size is 260 bytes.

The maximum recommended ccTalk data size is 252 bytes such than 255 bytes follow the size byte, giving a maximum packet size of 257 bytes. This allows a 8-bit receive pointer to be used when storing message data - handy in microcontroller applications with very little RAM.

The maximum likely ccTalk data size is a little over 128 bytes. This is often used when splitting up large blocks of data into ccTalk message packets. Sending a block number or address along with 128 bytes of data is a nice binary number for filling flash memory etc.

It is not always necessary to store the entire ccTalk data message before processing it. The message can be received and 'thrown away' or handled 'on-the-fly'.

2.29 How do I know when a new ccTalk message begins ?

This problem has to be dealt with by all ccTalk nodes. In a stream of data bytes how do you know when one message ends and another begins ? The answer is by byte counting. Every ccTalk message consists of an address byte followed by a size byte. After the size byte another '3 + size' bytes follow before the next message begins. Every ccTalk node must receive and count all bytes, even if the message is not addressed to them. Fortunately, the processing resource to do this is very small and it can be done as a 'background task'. Another problem arises where the start of a ccTalk message is unknown or incorrect in the first place - the problem of message re-synchronisation. To solve this ccTalk relies on the use of a 'data timeout'. Since gaps between messages will be far higher than the gaps between bytes in a ccTalk message packet, this longer time can be used to reset the receive pointers and re-synchronise the address byte detection. If there is a gap of more than 50ms between incoming bytes, it can be assumed a new message is starting. This process is also essential in dealing with electrical noise which could disrupt the incoming byte stream and force a bad checksum.

2.30 How do I know which version of ccTalk to use ?

At this moment in time there is really only one version of the ccTalk specification. There are various options such as encryption on bill validators but these are now well established. Future revisions of the specification have been broadly compatible with older revisions and no issues have so far arisen. If a peripheral doesn't support a particular ccTalk command then there is no response and alternative commands or actions can be taken by the host machine.

The version of ccTalk that a peripheral supports can be requested with command header 4. Three bytes are returned. The first byte is the level and the second 2 bytes refer to the ccTalk generic specification revision. The level is used for minor changes / bug fixes (as determined by the peripheral manufacturer) and the revision for protocol conformance.

2.31 Other serial protocols seem more sophisticated. Why is this ?

There are various features which have been deliberately left out of ccTalk to simplify the implementation and to lower the cost. There is no 'hot plugging' of ccTalk peripherals. If you add a ccTalk peripheral to a powered bus then the host machine has no way of knowing this has been done. Likewise if a ccTalk peripheral is removed from the bus, the next command to it from the host will fail. The host may decide the peripheral is missing or faulty - it has no way of knowing. Also, the use of ccTalk is restricted to 'single master' applications. If there is more than one ccTalk master on the bus then message packets will collide and the protocol becomes very inefficient. The addition of a 'Busy' line would help but as virtually all applications in the Money Processing Industry are single master, and need to be for security reasons, the protocol has been biased this way accordingly.

2.32 Is there a user group for ccTalk ?

Contact Money Controls for the latest information. An initial UK group was set up in 1999 to promote ccTalk use throughout the industry.

2.33 Do I send decimal, hexadecimal or ASCII values in ccTalk ?

This can cause confusion in many serial protocols and is a common source of error. In the product specifications, any byte values which are sent to the peripheral in an example such as [0] or [1] means the DECIMAL VALUE, not the ASCII representation of this number. If the values are in hexadecimal, this will be made clear.

Hexadecimal numbers (hex numbers) when applied to bytes are up to 2 characters long and may include the characters A to F as well as the digits 0 to 9.

e.g. 12, 5F, AA, 9C and 5 are all hex numbers.

In C they are written as 0x12, 0x5F, 0xAA, 0x9C and 0x05.
In Visual Basic they are written as &H12, &H5F, &HAA, &H9C and &H5.

A value such as 12 is ambiguous because it could be 12 decimal or 12 hexadecimal (= 18 decimal).

An ASCII representation of a number is totally different to the number itself. The ASCII code for 0 is 48 decimal. The ASCII code for 1 is 49 decimal. They increase in sequence.

ASCII codes are normally enclosed in single or double quotes such as ‘A’ or “A”.

So a byte sequence [‘1’] [‘2’] [‘3’] would be in ASCII.

2.34 Why are there so many approved ccTalk connector types ?

Ideally there would be just a single ccTalk connector type but practical realities have meant that different connectors have ‘evolved’ to suit different applications over the years. Part of the standardisation process within ccTalk is to reduce the number of options available.

At the moment coin acceptors and bill validators should use a 10-way, dual row, mechanically-keyed, 0.1 inch pin header and serial hoppers should use a 10-way, single row, mechanically-keyed, 0.1inch pin header. There is an option on 3.5inch coin acceptors to fit the smaller 4-way JST connector where PCB space is severely restricted.

2.35 I don't understand MDCES commands. Can you explain ?

MDCES stands for 'Multi-Drop Command Extension Set'. A special subset of commands was included in ccTalk to help with address resolution. For most applications the address of ccTalk peripherals is pre-determined and no issues arise. The coin acceptor is on address 2, the hopper on address 3 and the bill validator on address 40. But suppose we have 2 coin acceptors, 4 hoppers and 2 bill validators on the same bus ? They must all have unique addresses to work. The address of ccTalk peripherals can be stored in RAM or EEPROM to allow dynamic addressing. The MDCES commands should only be used where the network configuration is unknown as the commands are slow to execute and randomising addresses can result in further clashes which then have to be resolved again. As the likelihood of having more than one hopper connected to a machine bus is very high, hopper addresses can be changed on the connector wiring harness which avoids the need for address resolution.

Header 253, Address poll

This command is used with the broadcast address to find out which devices are connected to the bus. Each peripheral responds with its address delayed by a proportionate time. The return packet is a single byte, breaking the normal ccTalk packet rules. Devices with the same address could clash producing spurious addresses.

Header 252, Address clash

This command is used to verify a specific peripheral address. Each peripheral responds with its address delayed by a random amount of time. Devices with the same address have a good chance of being resolved.

Header 251, Address change

This command is used to change a peripheral address to the value specified. The next command to this peripheral should be to the new address.

Header 250, Address random

This command is the solution to any kind of clash problem discovered with the previous commands. All peripheral addresses are randomised (by sending the broadcast address) in the hope that they will all become unique. There is a good chance of this happening where the total number of peripherals is small - usually less than 10.

Peripherals will not randomise their addresses to 0 or 1.

3. Host Machine Manufacturer Questions

3.1 How do I connect a PC to a ccTalk peripheral ?

A ccTalk device cannot be connected directly to a PC serial port because the single bi-directional data lines need to be split into TX and RX lines, and the data voltages need to be changed from TTL to RS232 levels. This can be done with a RS232 converter chip, 2 transistors and a diode. A circuit diagram is provided in the generic specification.

It is possible to manufacture a device which uses the ccTalk protocol but with a RS232 connector fitted for direct connection to a PC.

3.2 Is there a standard ccTalk 'driver' I can use ?

Money Controls does not currently issue any software drivers to use on a PC-based platform. It must be said that the requirement for drivers is minimal as the ability to transmit and receive bytes through the serial port is fully supported in C and Visual Basic development environments, and the formation of ccTalk message packets is a relatively simple task for a software engineer.

3.3 Is it possible to have 2 ccTalk masters on the same bus ?

This is theoretically possible in the base protocol as the source address can be used to discover which master issued the command. However, multi-master ccTalk systems are strongly discouraged as the probabilities of message clashes are high and data integrity cannot be guaranteed. It is possible to add additional hardware to switch masters in and out of the bus, or to provide a 'busy' line, but the remit of this falls outside the published ccTalk specification. Note that when encryption is used, the source address field is mapped into a CRC checksum and so this method of determining the master is lost. The master is always assumed to be address 1.

3.4 Why is the original PNP ccTalk interface circuit now obsolete ?

An early ccTalk interface circuit used a PNP transistor for receiving data, with the base connected to the ccTalk data line. It soon became apparent that the serial hoppers were dragging the data line down to 4.5V which was turning on the PNP transistor whilst in idle. The PNP was therefore replaced by a diode and this has fixed the problem.

3.5 If I use the broadcast address, all the responses clash. So why use it ?

Good question but there is a reason. The broadcast address was originally included in the protocol to allow the MDCES command, 'Address poll', to be sent to all devices simultaneously. Responses to this command are single bytes and staggered in time so the chance of a collision is much reduced. Another use is when a single peripheral is connected to a diagnostic terminal and you are unsure what the address is. By using the broadcast address you can be sure the peripheral will reply. This turns out to be incredibly useful in practice.

The host machine can of course ignore all receive bytes from a broadcast command for a fixed period of time. This works well when the message is an action rather than a request for data.

3.6 Can I run a ccTalk serial cable between machines ?

This is possible but screened cable would be recommended to reduce noise. The standard ccTalk interface electronics are only designed for short distance hook-up of peripherals within a machine rather than between machines.

3.7 I use Linux. Is that a problem ?

The use of ccTalk is not restricted in any way by the choice of operating system. The only requirement when developing host software is the ability to control and access a UART.

3.8 My company doesn't have electronic engineering resource. How do I use ccTalk ?

For companies developing software on a PC, Money Controls can supply a RS232 to ccTalk interface box. The only task remaining is to write the application and ccTalk control software.

3.9 My company does not have software engineering resource. How do I use ccTalk ?

Since ccTalk is a serial protocol, software resource will be required to make it work with your application. Sub-contracting is probably the only option, although given the nature of the protocol this should be a relatively simple task.

3.10 How do I know what coins and bills are available to me in a peripheral ?

There are ccTalk commands available which list the coins and bills that can be accepted by the peripheral. This removes the need for a fixed look-up table.

Header 184, Request coin id

Header 157, Request bill id

The host machine should perform an 'enumeration' of coins and bills during the power-up initialisation routine. For each programmed channel or position within the validator, which corresponds to the serial credit code returned during polling, the host machine reads out the associated coin or bill identification string. These are then stored in a 'RAM table' which are available for look-up as credit codes are generated.

3.11 I have local echo. Is that correct ?

The ccTalk protocol uses a bi-directional data line. Any transmitted data will appear on the data bus and therefore most likely immediately on the receive port of the transmitting device. This is indeed the case with the standard ccTalk PC interface circuit (see circuit 4 in the generic specification). The software can disable the receive port while transmitting, or clear the receive buffer immediately afterwards, but the most elegant solution is to index into the receive buffer the number of transmitted bytes in order to find the reply packet. So if the ccTalk master sends a 6 byte message packet it will read the reply packet at byte position 7 since the first 6 bytes will be a straight copy of the transmitted packet. If this is not the case then it can be assumed there is a catastrophic comms failure.

When the slave device replies, it too will have local echo and will have the host reply packet in its receive buffer. This will be addressed to the master device with return command header zero. This message should therefore be 'ignored' by the slave firmware - wrong address.

The existence of local echo is a common occurrence in many communication systems and can easily be allowed for once you know it is happening.

4. Peripheral Manufacturer Questions

4.1 What is the minimum hardware I need to run ccTalk ?

ccTalk is one of the simplest asynchronous protocols that it is possible to use. Although it is multi-drop and supports variable message lengths, there is no 9th address bit or wake-up bit in the character frame. The need for parity bits has also been removed. Although it is possible to write a software UART on a microcontroller to support ccTalk, the use of a hardware UART eliminates many of the potential timing problems that can occur. A typical microcontroller requirement for ccTalk would be a 8-bit core with a 8/16-bit timer, hardware UART and 2K ROM. Application code would be additional to this of course. The amount of RAM depends on whether messages are fully buffered in the transmit and receive paths or whether they are processed 'on-the-fly'. Typically anything from a few tens of bytes to a few hundred bytes of RAM.

4.2 How do I create a ccTalk product ?

Any peripheral manufactured with an interface conforming to the ccTalk generic specification can be referred to as a ccTalk product. Commands exist to identify the manufacturer, product name, build code, software revision, ROM checksum, serial number and manufacturing date. All these fields can be filled in appropriately.

4.3 I want to add some 'secret' ccTalk commands. How do I do it ?

Header 255 is the 'Factory set-up and test' command and can be used by any manufacturer for internal functions. No details of these functions need ever be published. If thought necessary, various security mechanisms can be put in place to protect this command from unauthorised use. One of the simplest is a PIN number. Although only a single ccTalk header is defined for this function, the first byte of the data payload can be a sub-header, massively expanding the command set.

4.4 How do I report multiple fault codes ?

The ccTalk protocol currently only supports priority fault code reporting. The fault code is returned in response to ccTalk header 232, 'Perform self-check'. So if fault codes A, B, and C suddenly develop on a product, which is unfortunate in the extreme, fault code A is returned until fixed, then B and finally C.

4.5 Can I implement ccTalk on a Microchip PIC microcontroller ?

Yes - even a small PIC microcontroller can be used for a ccTalk peripheral. The Mk 1 serial hopper from Money Controls used a PIC12C671 for the entire application. This 8 pin device has 1K(word) ROM, 128 bytes RAM and an internal RC oscillator. 28 ccTalk command headers were implemented.

4.6 Is there a ccTalk logo I can use on my products ?

Not yet but watch this space.

4.7 How does polling work with a fast coin acceptor in gaming ?

A coin acceptor for gaming may be capable of accepting 20 coins per second in short bursts. However, the ccTalk host controller may only be polling the coin acceptor once per second. So how can we not lose any credit information ? The simple answer is buffering. The ccTalk command header 229, 'Read buffered credit or error codes', can read up to 5 new credits. So in 4 seconds the host machine can discover all the coins put down in a second. This relies on peripheral 'double-buffering'. There is a 10 byte transmit buffer containing credit events and a much deeper credit stack allowing continuous coin feeding. Eventually the stack will run out of course and the coin acceptor would have to self-inhibit for a few seconds.

There are a number of complications to this ideal scenario.

The first is that if 5 new credits are transmitted at each request and some return communication error means the host has to retry the command, it would lose that credit information. The event counter would indicate 5 missing credits but no details of those credits could be obtained. For this reason it is recommended that only 2 new credit events are added to the transmit buffer at each poll. This allows a single retry to be made without loss of data. To compensate for this reduction in data transfer it is recommended the coin acceptor is polled every 200ms. So 10 coin credits can obtained every second, and 20 coins would be handled in 2 seconds, twice the coin insertion time.

The second is power loss. The disadvantages of a deep stack are that if power is lost then the entire stack is lost unless it is backed up to non-volatile memory. This is true though of any serial protocol where the transfer of data is potentially slower than the coin entry speed. Also, any serial protocol where the transfer of data is slower than the power supply fall time can lose a single credit. So it is up to the system designer to ensure that vulnerability to power loss is at a minimum, whether it is through the use of battery-backed RAM and / or system boards, EEPROM memory or minimum polling requirements.

4.8 Does ccTalk support remote download of coins and bills ?

Yes - support is provided. The bill validator commands are in the public header section while the coin acceptor commands are currently in the application specific section.

Each peripheral manufacturer will have a very different method of downloading new coin sets or bill tables into a peripheral and so it is pointless standardising the process beyond a generic block transfer of data. Variable length data from a few bytes to millions of bytes is supported. The format of the data has been left to each manufacturer. The 'begin' and 'finish' commands have been included to allow a convenient method of invoking a FLASH memory erase and program cycle.

Header 143, Begin bill table upgrade
Header 144, Upload bill tables
Header 142, Finish bill table upgrade

4.9 What power can I source over a ccTalk serial bus ?

There is currently no specification for the power available over ccTalk to each peripheral. It is assumed that the host machine power supply is rated sufficiently to drive all peripherals that may be attached to the bus. Unlike USB which places a limit of 0.5A at 5V and which supports random hot-plugging of peripherals, all ccTalk networks will be pre-determined and the power calculation will have been done at the system design stage. It is likely that a machine manufacturer will place a requirement on the peripheral manufacturer when approving new devices.

Coin hoppers are particularly 'hungry devices'. The serial hoppers manufactured by Money Controls require 3A peak at +24V. We recommend that only a single coin type is dispensed at a time, to reduce the requirement for heavy gauge bus wires and more expensive connector types. It also reduces the amount of 'ground shift' that can occur when all motors are running. The standard electrical interface circuit of ccTalk is open-collector and is sensitive to these shifts.

Each peripheral manufacturer has the option of fitting a separate power supply connector to reduce the loading on the main ccTalk bus. In this case only the 0V and data line need to be connected to the bus with the supply voltage coming from the auxiliary connector.

4.10 How do I register our company name ?

The ccTalk command 'Request manufacturer id', header 246, requests the ASCII identification string from the peripheral. There are currently 2 formats listed in the generic specification - full names and abbreviated names. Abbreviated names consist of 3, unique, upper case characters. Abbreviated names have been recommended for use on bill validators. If you wish to have a company listed in the generic specification and you are a manufacturer of ccTalk peripheral equipment then please contact Money Controls. Host machines will vary in the action they will take on discovering an unknown manufacturer. Some machines will only operate with a previously approved company and product identifier. Others will ignore this technicality and continue operation with the generic command set.

4.11 I haven't got time to respond on ccTalk. How do I solve this problem ?

The environment in which ccTalk operates is a master-polled one. The master will poll round each peripheral in turn, usually looking for new events, and it is the duty of each peripheral to respond promptly. If a peripheral does not respond immediately then it is preventing other devices from being polled and eventually the bus will become unusable. The options for a peripheral which is 'busy' on other tasks are as follows…

a) Send a 'no event' reply.

The peripheral software should be capable of handling message responses as a 'background' task. It should at least be able to respond to an event poll with the same response as last time and as the event counter will be unchanged this indicates to the host that no new events have occurred - even if they really have. The effect is to delay the reporting of events until a more convenient point in the peripheral software. This method does not require special action on the host side and system polling can continue as normal. To generalise this approach further, ccTalk commands should be split into 'immediate response' and 'initialisation and diagnostic' commands. Only the immediate response commands will be handled with the peripheral in a 'cash in / cash out operating state', the others can be dealt with at switch-on or when using diagnostic routines. The immediate response commands are not listed in the generic specification but are typically simple polls, inhibit modifying and event polling. Also any commands used to switch encryption keys.

b) Send a BUSY response - header 6.

The normal command return header is zero but if it is 6 then this indicates to the host machine that the peripheral is too busy to reply. Use of this method is discouraged - consider making serial communications a higher priority task.

c) Do not respond.

Obviously this option is always open to the peripheral but the host has no choice but to wait for a 'timeout' value and then retry or move on to the next peripheral. The host will want the timeout value to be as short as possible. It also has no idea which of the following conditions has occurred…

- Peripheral too busy to respond
- Peripheral removed from bus
- Power lost from peripheral
- Peripheral developed a fault
- Message corrupted (bad checksum)
- Incorrect address or encryption settings

This manual is intended only to assist the reader in the use of this product and therefore Money Controls shall not be liable for any loss or damage whatsoever arising form the use of any information or particulars in, or any incorrect use of the product. Money Controls reserve the right to change product specifications on any item without prior notice