Part 4 - Contents

1. ccTalk FAQ - Frequently Asked Questions

1.1 General Questions

1.1.1. Where does the name ‘ccTalk’ come from ?

The protocol was developed at Coin Controls before the company changed its name. Hence **c**oin-**c**ontrols-**Talk**. We are now Crane Payment Solutions - Money Controls.

1.1.2. Which is correct - ‘cctalk’ or ‘ccTalk’ ?

The marketing brand name ‘ccTalk’ has now replaced the original engineering name ‘cctalk’ in most of the glossy brochures. Older specification documents may still use ‘cctalk’. No other case formats are officially recognised.

The name ‘ccTalk’ should be used by all manufacturers of new equipment on labels and in technical manuals.

1.1.3. 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.

1.1.4. What is ccTalk and who uses it ?

ccTalk is a serial protocol for use in the *Money Transaction 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.

1.1.5. Where do I get information on ccTalk ?

The most important document is the ‘ccTalk Serial Communication Protocol - Generic Specification’ (of which this FAQ is now part) 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 Crane Payment Solutions - Money Controls or via the ccTalk web site at www.cctalk.org.

1.1.6. Who is the controlling authority for ccTalk ?

It is currently with Crane Payment Solutions - 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

1.1.7. 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.

1.1.8. What software support can you provide ?

Crane Payment Solutions - Money Controls is primarily a hardware manufacturer and does not provide a suite of tools for software development. We can provide specification documents and some limited 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.

1.1.9. 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 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 (half-duplex operation) and the mark / space voltages of ±12V have been changed to 0V / +5V.

1.1.10. 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.

1.1.11. 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.

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

Hindsight is a wonderful thing ! The ccTalk headers were created from 255 downwards 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.

1.1.13. 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 operate in secure ccTalk mode with encryption on every command.

1.1.14. Has anybody written a ccTalk protocol analyser ?

Crane Payment Solutions - Money Controls has a very basic software package called 'ccAnalyse'. 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.

1.1.15. 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.

Note that the year 2005 saw the beginning of a number of specialist ccTalk applications running over a USB link with a virtual COM port driver. This allows very high baud rates in excess of 1Mbps to be used. See Section 5.

1.1.16. 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 a maximum of 10 to 20 devices.

1.1.17. 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.

1.1.18. 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.

1.1.19. 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 in a conventional multi-drop system however as there would be a 9600 baud bottleneck on existing peripherals.

USB to RS232 converter cables are now widely available and relatively inexpensive. When the driver is installed on a PC, a virtual COM port should be available for use with ccTalk. See Section 5 for more information.

1.1.20. Is there a ccTalk test house or approval facility ?

Not at this point in time. We operate a process of self-certification.

1.1.21. When was ccTalk 'invented' ?

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

1.1.22. 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.

1.1.23. 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 !

1.1.24. Is there a web site for ccTalk ?

Yes - visit www.ccTalk.org

1.1.25. Is there a user group for ccTalk ?

Contact Crane Payment Solutions - Money Controls for the latest information. An initial UK group was set up in 1999 to promote ccTalk use throughout the industry but now any discussion is carried out by email.

1.1.26. 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 points…

- No response is received from the slave (the standard ccTalk error condition)
- A NAK message is received from the slave (used with hoppers)
- There is a low-level RS232 framing error (stop bit invalid)
- There is a data underrun or overrun based on the ccTalk 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 miscounted. 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.

1.1.27. 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.

1.1.28. How do I go about obtaining ccTalk encryption documents?

The encryption documents are obviously sensitive and will not be made available in the public domain. If you require the documents for serial hoppers or bill validators then contact Crane Payment Solutions - 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.

1.1.29. 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 Transaction Industry are single master, and need to be for security reasons, the protocol has been biased this way accordingly.

1.1.30. 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 (= 256 x 3 + 0).

1.1.31. 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 Part 3 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

1.1.32. 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 is possible to have 255 data bytes such that the maximum packet size is 260 bytes.

The maximum implemented ccTalk data size is often just over 128 bytes. This is used for certain commands 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'.

1.1.33. 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.

1.1.34. 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.

1.1.35. 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 arisen so far. 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.

1.1.36. 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.

1.2 Host Machine Manufacturer Questions

1.2.1. 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 Part 3 of 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.

1.2.2. Is there a standard ccTalk ‘driver’ I can use ?

Crane Payment Solutions - 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.

1.2.3. 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.

1.2.4. 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.

1.2.5. 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. Also, if you wish to randomise all addresses on the ccTalk bus then you can send the MDCES command 'Address random' and ignore the return 'garbage' data (clashing ACKs) for a set time. This approach works for all non-critical commands where return data is not needed.

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.

1.2.6. 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.

1.2.7. 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.

1.2.8. My company does not have electronic engineering resource. How do I use ccTalk ?

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

1.2.9. 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.

1.2.10. 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.

1.2.11. 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 Part 3 of 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.

1.3 Peripheral Manufacturer Questions

1.3.1. 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.

1.3.2. 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.

1.3.3. I need some extra ccTalk commands. How do I do it ?

Existing commands should be used wherever possible as this massively simplifies machine software and helps the process of global standardisation. However, if none of the new features you wish to support are a 'good fit' with existing command and data fields, then headers 20 to 99 are available for custom use. No agreement is necessary within the industry as they are all regarded as application and manufacturer specific. For a host machine to make use of these extra commands, it should identify the product first to see if they are supported. Header 99 may be used on Product A from Manufacturer A to do Task A, but the same header may be used on Product B from Manufacturer B to do Task B.

It is essential that if these extra commands are regarded as generally useful throughout the industry that they are promoted to the public header section of the ccTalk command set as soon as possible. This prevents the widespread distribution of custom machine software. Please contact Crane Payment Solutions - Money Controls if you wish to expand the ccTalk command set in this manner.

1.3.4. 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, expanding the command set in any way required.

1.3.5. 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 Crane Payment Solutions - 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.

1.3.6. 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

1.3.7. 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.

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

Not yet but this will change if a new certification process is adopted.

1.3.9. 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

1.3.10. 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 Crane Payment Solutions - 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.

1.3.11. 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.

1.3.12. 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.

2. ccTalk PDR - Peripheral Design Rules

Now that ccTalk has been adopted by many peripheral manufacturers throughout the world, we are looking at tightening up some areas of the specification which have been open to interpretation. This will lead in future to better interoperability. Please regard these as being in force on new product designs.

2.1 Coin Acceptors

2.1.1. Rule 1 - Command Set

A minimum set of commands must be implemented on coin acceptors. Refer to Appendix 13, Minimum Acceptable Implementations, in Part 3 of the generic specification.

2.1.2. Rule 2 - Credit Poll Timeout

The coin acceptor should stop accepting coins if there is fault with the host machine. The integrity of the host to device link is established through credit polling. If the host machine fails to poll the coin acceptor at least every **1 second** then the coin acceptor should self-inhibit. When the host machine resumes polling, this self-inhibit should be removed automatically by the coin acceptor. The host machine does not need to explicitly re-enable the coin acceptor.

2.2 Bill Validators

2.2.1. Rule 1 - Command Set

A minimum set of commands must be implemented on bill validators. Refer to Appendix 13, Minimum Acceptable Implementations, in Part 3 of the generic specification.

2.2.2. Rule 2 - Credit Poll Timeout

The bill validator should stop accepting bills if there is fault with the host machine. The integrity of the host to device link is established through credit polling. If the host machine fails to poll the bill validator at least every **5 seconds** then the bill validator should self-inhibit and insert a ‘Master inhibit active’ event on the event code stack. Any illumination on the front of the bill validator should indicate out-of-service. When the host machine resumes polling, it should see the master inhibit event and re-enable the bill validator with the ‘Modify master inhibit status’ command. The illumination on the front of the bill validator should then indicate that bills can be inserted.

2.2.3. Rule 3 - Escrow Operation

If a bill validator supports escrow mode then it should be enabled by default at power-up. It should not be necessary to select escrow operation with the 'Modify bill operating mode' command. To test whether escrow mode is available then the host machine can use the 'Request bill operating mode' command.

2.3 Payouts

2.3.1. Rule 1 - Command Set

A minimum set of commands must be implemented on payouts. Refer to Appendix 13, Minimum Acceptable Implementations, in Part 3 of the generic specification.

2.3.2. Rule 2 - Event Counter

The event counter is returned by the 'Dispense hopper coins' command and during polling with the 'Request hopper status' command. If a dispense command is sent from the host machine to the hopper and it is corrupted by noise then the hopper does not see a valid ccTalk message and there is no reply. No action is taken by the hopper and its operating state is unchanged. The host machine can check the event counter and re-issue the dispense command. However, if the hopper receives the command without error but the reply back to the host machine is corrupted, then the host was either sent an event counter by way of an acknowledgement or a NAK packet to say the dispense operation was refused. It may be possible to immediately request the hopper status to find out whether coins are actually being dispensed but if the hopper has finished then the only way of knowing what happened might be to read the dispense counter and compare it to the value before the command was sent. For this reason it is preferable **to only increment the event counter if the dispense command was accepted and coins paid** (or tried to be paid in the event the hopper was empty). The next encryption key should be generated by the hopper regardless of whether the command was ACK'd or NAK'd. The host machine therefore always needs to request a cipher key prior to a dispense command being sent.

3. ccTalk 'Combo Devices' White Paper

One question that can arise on ccTalk is 'Can we combine several peripherals into one combo device and operate it through a single ccTalk connector ?' The schematic for this is shown below.

Figure 1

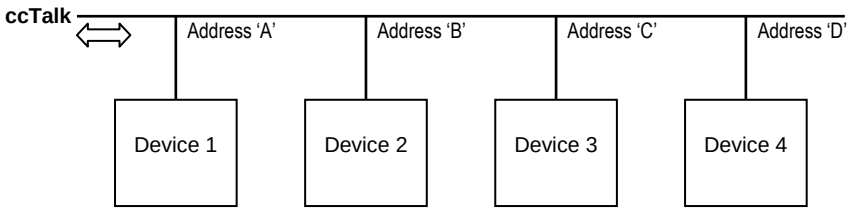


Figure 2

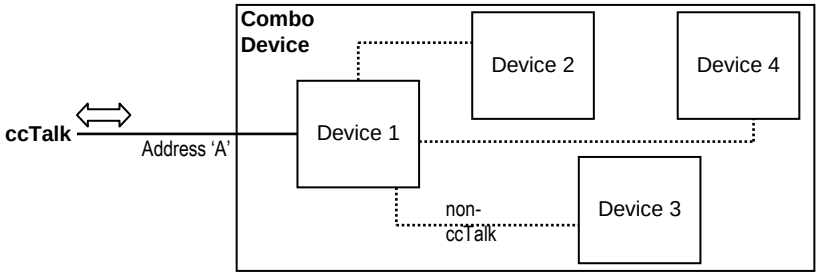


Figure 1 shows a classic ccTalk bus with daisy-chained peripherals. Each peripheral exists in its own right on the ccTalk bus and has a unique address.

Figure 2 shows a combo device perhaps consisting of a coin acceptor and three hoppers or a coin acceptor, bill validator, card reader and printer. Physically they exist as a single unit with a single ccTalk connector. Device 1 may consist of a single MPU board and the other devices could be 'dumb' peripherals with electronics but no firmware. The link between device 1 and the other devices may or may not be serial. If serial then it is probably not ccTalk since the topology would then be identical to Figure 1, assuming the bus is common. However, an isolated bus would allow split master operation which is not a case considered here.

The main problem is how do we operate functionally different devices through a single, and by implication single ccTalk address, connector ?

Consider the following commands and technologies…

- Header 245, Request equipment category id
- Header 244, Request product code
- Header 192, Request build code
- Header 242, Request serial number
- Header 241, Request software revision
- Header 4, Request comms revision
- BNV encryption
- Hopper encryption
- Headers 250-253, MDCES commands

We could create a new category of device such as 'Changer'. This would have a single ccTalk address and a unique identification.

For example…
Request equipment category = 'Changer'
Request product code = 'MagicPayDay'
Request build code = 'Standard'
Request serial number = 12,345,678
Request software revision = 'SP-1.00'
Request comms revision = 1.4.6

The problem now is how do we address individual devices within the combo device ? For instance, how do we dispense coins from hopper 1 rather than hopper 2 ? How do we perform diagnostics on, and differentiate fault codes between, a coin acceptor and perhaps a bill validator ? The 'Modify inhibit status' command is the same for coin acceptors and bill validators so how can we 'steer' it towards one device and away from another ? If we wish to use unencrypted ccTalk on the hoppers but encrypted ccTalk on the bill validator then how can that work ?

Consider now solutions for a Changer comprising a single coin acceptor, bill validator and multiple hoppers.

3.1 Solution 1 : Manage As Single Device

We treat the entire system as a single device with a single identity. This identity is 'Changer'. We now need to add additional bytes to many ccTalk commands to operate the devices independently. As we are not allowed to modify existing ccTalk commands this means creating lots of new ones.

For instance, the Dispense hopper coins command would become…
Transmitted data : [hopper no.] <encryption data> [no. of coins]

The Request hopper status command would become…
Transmitted data : [hopper no.]

The Perform self-check command would become…
Transmitted data : [coin acceptor or bill validator]

If the encryption layer is switched on then all devices would be encrypted as all commands go to one address. The hopper encryption key could be unique to each hopper as the 'Request cipher key' command would have a [hopper no.] byte added to it.

The MDCES commands would operate as normal on the single ccTalk address.

3.2 Solution 2 : Manage as Multiple Devices on Single Address

The trick here would be to add a new ccTalk command which specified a sub-address.

Header XXX : Forward ccTalk packets

```
Transmitted data : [ mode ] [ forward address ]
Received data : ACK

[ mode ]
0 - no packet forwarding
1 - forward next packet only
2 - forward all packets until cancelled with 0

[ forward address ]
0 to 255 - sub-address of ccTalk peripheral
```

This would allow existing ccTalk commands to be forwarded to the specified sub-address without changing them. Each virtual device would have a unique sub-address.

Assume the Changer is on address 200.

To dispense coins from hopper 1…

```
TX : [ 200 ] [ 2 ] [ 1 ] [ XXX ] [ 1 ] [ 3 ] [ CHK ] - next command sent to sub-address 3
RX : [ 1 ] [ 0 ] [ 200 ] [ 0 ] [ CHK ]

TX : [ 200 ] [ 9 ] [ 1 ] [ 167 ] [ sec1 ] [ sec2 ] [ sec3 ] [ sec4 ] [ sec5 ] [ sec6 ]
[ sec7 ][ sec8 ] [ noCoins ] [ CHK ]
RX : [ 1 ] [ 1 ] [ 200 ] [ 0 ] [ events ] [ CHK ]
```

To dispense coin from hopper 2…

```
TX : [ 200 ] [ 2 ] [ 1 ] [ XXX ] [ 1 ] [ 4 ] [ CHK ] - next command sent to sub-address 4
RX : [ 1 ] [ 0 ] [ 200 ] [ 0 ] [ CHK ]

TX : [ 200 ] [ 9 ] [ 1 ] [ 167 ] [ sec1 ] [ sec2 ] [ sec3 ] [ sec4 ] [ sec5 ] [ sec6 ]
[ sec7 ][ sec8 ] [ noCoins ] [ CHK ]
RX : [ 1 ] [ 1 ] [ 200 ] [ 0 ] [ events ] [ CHK ]
```

So all host communication is to address 200 but commands could be sent to virtual peripherals on sub-addresses if required. Each virtual peripheral could have its own serial number and identity.

Rather than create a new category called 'Changer' on address 200 we could put the coin acceptor on address 2 and have the hoppers addressable via the coin acceptor. So all commands (even those intended for the hoppers) would go to address 2. This means a new category is no longer required.

If the encryption layer is switched on then all devices would be encrypted as all commands go to one address. The hopper encryption key could be unique to each hopper as the Request cipher key command is forwarded to the correct sub-address.

The MDCES commands would operate as normal on the single ccTalk address. It would be possible to forward MDCES commands to the sub-addresses but this seems unnecessarily complicated and could be prevented. For instance, if we forward a broadcast address to all sub-addresses then do real and virtual devices all reply ?

3.3 Solution 3 : Manage as Multiple Devices on Multiple Addresses

This requires Device 1 to have a virtual address handler. It will check for an address match on a range of addresses, e.g. 2 to 5 and 40. Address 2 commands will be handled by the coin acceptor firmware module and addresses 3, 4 and 5 will be handled by the corresponding hopper firmware module. Address 40 will be handled by the bill validator firmware module.

From a host software point of view, operation is identical to Figure 1. The coin acceptor, bill validator and hoppers could have their own serial numbers and identity if required.

If the encryption layer is required on any of the devices then Device 1 can decide by looking at the address byte whether to decrypt the incoming message or not (the address byte and no. of data bytes are never encrypted). The hopper encryption key could be unique to each hopper.

The MDCES commands could be supported on virtual devices if software is written to handle the additional complexity. For instance an ‘Address poll’ on the broadcast address could return all virtual addresses. An ‘Address random’ on the broadcast address could change all of the virtual addresses to new ones. The alternative is to prevent any address changes from occurring.

3.4 Summary

The table below summaries what has been discussed.

Solution	Addresses	Additional ccTalk commands	Encryption Layer
1	Single	Many	All or Nothing
2	Single	One	All or Nothing
3	Multiple	None	Selectable

3.5 Conclusion

Solution 1 requires a new product category and many additional ccTalk commands. Although the single device concept is compact and elegant, the ccTalk command structure becomes inconsistent, ambiguous and hard to follow. It would *destroy* the protocol.

Solution 2 is again elegant but there is now an overhead in the protocol as all commands to hoppers, for example, have to be prefixed by a sub-address mask. This will complicate host software drivers and increase the possibility of a 'system hang' if the address switching goes out of sync. This is probably best avoided if we can.

Solution 3 is easy to comprehend as the combo device behaves as if it was individual devices attached to the ccTalk bus. There are no changes or additions to the existing ccTalk command structure and any driver software already written should still work.

So Solution 3 is the best option requiring a virtual address handler to be written in the peripheral device. The overhead for this is very low and the software is easy to implement.

Note there is a new ccTalk header 105, 'Data stream', which attempts to solve some of these issues for memory transfer using a peripheral identifier byte in the packet structure.

4. ccTalk RFC (Request For Change)

The following changes are being investigated for future integration into the ccTalk specification. Representations are invited from all interested parties.

New RFCs may be submitted by member companies to abarson@moneycontrols.com

4.1 Addition of lower-case letters to coin & note mint issue

Filing Code	RFC/001
Filing Date	28/12/05
Compatibility Issues	None expected but see comments
Current Status	Rejected

The letter codes A-Z are used to differentiate between coins and notes of the same currency and value.

See Appendix 3 – Coin Types and Coin Values.

See Appendix 15 – Bill Types and Bill Values

26 issue codes were originally thought to be plentiful to deal with new and old money and a number of minting variations. The code is of no interest to many host machine applications which deal principally with credit 'value' for the purposes of determining a machine game or vend level. However, when inhibiting specific coins or bills in circulation, or downloading new coin and bill currency sets, it becomes important to identify the monetary unit precisely.

Some currencies such as British banknotes have reached letter Z already as there are printings by Scottish and Northern Ireland banks included. A number of countries and banks share the same 'GB' ISO code.

It is proposed to add 26 further codes using the lowercase letters a-z.

The order of assignment will be uppercase first then lowercase i.e. A, B, C… Z, a, b, c… z.

The single character format is retained to maximise compatibility with existing software. No change will be seen until the 26 uppercase options are exhausted.

Comments
Coin and bill names are often used in filenames and folder names. It may be the case that a file system does not distinguish between upper case and lower case letters. So a GB0020A.xxx file would be no different from GB0020a.xxx. At the ccTalk interface level this does not cause a problem but it requires work-arounds for supporting software. It would be preferable to add an extra issue character but for backwards compatibility this cannot be done with headers 184 and 157.

4.2 Addition of decimal point into the bill value field

Filing Code	RFC/002
Filing Date	28/12/05
Compatibility Issues	Minor problems could occur
Current Status	Under Consideration

Bill validators have a global scaling factor and decimal point for each currency. When a country has new and old notes in circulation with widely differing scaling it can be difficult to derive a common scaling factor which works for all notes.

Rather than change to a system whereby each individual note has its own scaling factor, it is proposed to allow decimal points to be used in the 4 character value field. This gives a greater value range than is possible now.

New codes could typically be… (preferred value codes are highlighted in green)

Value Code	Scaling Factor	Decimal Places	Result
.001	10,000	2	0.10
.002	10,000	2	0.20
.005	10,000	2	0.50
.010	10,000	2	1.00
.020	10,000	2	2.00
.025	10,000	2	2.50
.050	10,000	2	5.00
.100	10,000	2	10.00
.200	10,000	2	20.00
.250	10,000	2	25.00
.500	10,000	2	50.00
0.01	10,000	2	1.00
0.02	10,000	2	2.00
0.05	10,000	2	5.00
0.10	10,000	2	10.00
0.20	10,000	2	20.00
0.25	10,000	2	25.00
0.50	10,000	2	50.00
1.00	10,000	2	100.00
2.00	10,000	2	200.00
2.50	10,000	2	250.00
5.00	10,000	2	500.00
00.1	10,000	2	10.00
00.2	10,000	2	20.00
00.5	10,000	2	50.00
01.0	10,000	2	100.00
02.0	10,000	2	200.00
02.5	10,000	2	250.00
05.0	10,000	2	500.00
10.0	10,000	2	1,000.00
20.0	10,000	2	2,000.00
25.0	10,000	2	2,500.00
50.0	10,000	2	5,000.00
0001	10,000	2	100.00
0002	10,000	2	200.00

0005	10,000	2	500.00
0010	10,000	2	1,000.00
0020	10,000	2	2,000.00
0025	10,000	2	2,500.00
0050	10,000	2	5,000.00
0100	10,000	2	10,000.00
0200	10,000	2	20,000.00
0250	10,000	2	25,000.00
0500	10,000	2	50,000.00
1000	10,000	2	100,000.00
2000	10,000	2	200,000.00
2500	10,000	2	250,000.00
5000	10,000	2	500,000.00

The currency scaling factor of 10,000 could allow a note value of 500,000 giving a ratio of highest value note to lowest value note of 500,000 / .1 = 5 million which is considerably better than the current ratio of 5,000.

Comments

The only unknown is how existing machine software would react to a decimal point in the value field when it is expecting a whole number. The impact of this may have to be investigated on a case-by-case basis.

4.3 Poll Watchdog Event and Baud Rate Switching

Filing Code	RFC/003
Filing Date	23/10/06
Compatibility Issues	9600 baud is the default ?
Current Status	Discussion Topic
Posted	Jan Kaiser / Phoenix Mecano / Digital Elektronik GmbH

I have some ideas for improvement of the ccTalk standard. These are:

1. Coin Acceptor new Error Code 30 = Poll Watchdog was going active. That means coin acceptance was inhibited due to missing #229. This event is set on each occurrence, which can be the case only once after each #229. No coin is rejected. (Each inserted coin while poll watchdog is active (alerted) will create an inhibited coin event.)

Comments
The credit poll watchdog mechanism stops the coin acceptor accepting coins if the host machine 'dies' and stops polling the acceptor. Although we could generate events if this condition occurs and every time a coin is entered and inhibited, there seems little point as there is a host-side fault. There is some flexibility on how this feature is implemented by peripheral manufacturers; for instance, inhibited events could be generated.

2. Communication speed change header
A new header for changing the communication speed would be useful.
Two data bytes should be transmitted from the master:
data1: 0 = ask availability
1 = set speed 9600Baud (standard level)
2 = set speed 19200 Baud
data2: wait time, proposed format = data2*10ms
Answer of device:
If asked, one byte with set bit for each supported speed level. Bit 0 is kept cleared.
If set on broadcast address, none and speed change.
If set on device address, ACK and than speed change after sent complete.
Intended use:
All communication starts at 9600Baud standard speed. The master asks all devices for available speeds. If all devices on the bus support higher speeds the speed change command is issued by the master.
This should be done as a broadcast command. If the speed change is set individually the device ignores all communication for wait time, thus allowing other devices switching the speed.
An automated fallback seems useful. If 30s after wait time no header arrived or 10s after wait time 10 communication errors occurred (framing error indicating wrong speed) the speed should be set back to the standard level by each device on the bus automatically. The master can set the speed back too.
Could possibly used for fast block transfers (like update).
Limitations:
This is only useful if all devices support the header and higher speeds.

Comments

The ccTalk protocol is now being run at much faster speeds than the industry-standard 9600 baud. For instance, using ccTalk over USB we can easily achieve 1Mbps. We could have implanted a baud rate switching command as above but it seems most machines prefer to operate with a fixed baud rate and this is implemented in the peripheral as a factory configuration option or perhaps on a DIP switch. Also, all peripherals on the bus must operate at the same baud rate so if some support automatic switching and others don't there are going to be compatibility issues.

However, see new ccTalk header 113, 'Switch baud rate'.

3. Coin acceptor: Note on Hardware, USB tunnelling

The ccTalk protocol is suited for tunnelling via USB. This is useful for direct PC-connections, where the PC is the master. An USB bridge could be onboard of the coin acceptor or in an adaptor.

There are the following differences to the common ccTalk cable connections:

· specified standard connectors and cable are not used, instead use of standard USB cable, with connectors device-B or device-mini on the coin acceptor. (The coin acceptor is usually not bus powered, an additional power cable is required but not standardized. A DC power jack seems to be common.)

· no echo on USB cable (important for software on PC)

Comments

Agree that USB tunnelling is a very useful concept – see below. However, maximum compatibility with existing drivers is achieved by retaining the local echo feature. In other words, there is loop-back on the TX, even over USB.

4.4 Baud Rate Switching

Filing Code	RFC/004
Filing Date	12/11/07
Compatibility Issues	Change to 9600 baud rate ?
Current Status	Completed
Posted	Alex Pogossov, Microsystem Controls Pty Ltd

When using ccTalk we sometimes increase baud rate, mainly to speed up downloading and uploading configuration data, etc.

At the moment we are using a proprietary command for that. It works as follows:

The host sends one byte:
1 for 9600 bps;
2 for 19200 bps;
3 for 38400 bps;
4 for 57600 bps;
5 for 115200 bps;
(other values have no effect.)

A device would ACK the command, if the byte is in the range 1..5 or NACK it otherwise. The device would reply at the original speed and would switch to a new speed shortly after the stop bit of ACK's checksum has been transmitted.

On power-up or after reset all devices start at 9600 bps. Also, as a safety measure, if a new higher rate appears too unreliable, the host can force all the devices to the default 9600 bps by holding the ccTalk data line low for at least 50 ms.

If you find this approach useful for the future ccTalk protocol expansion, you might like to make such approach "official" in the next revision and reserve a special header for it.

If you are aware of some other way of baud rate control in ccTalk, which is likely to become a defacto standard, please let us know, and we might adopt it as well.

See new ccTalk header 113, ‘Switch baud rate’.

4.5 Product Spoofing

Filing Code	RFC/005
Filing Date	07/05/08
Compatibility Issues	The aim is backwards compatibility for new products
Current Status	Discussion Topic
Posted	Andy Barson, Money Controls

Some time after ccTalk was introduced it became apparent that many host machine applications were wired up (well, the software equivalent) to recognise only a specific type of ccTalk peripheral. This extended well beyond the equipment category id into other fields. From a security and approvals point of view this was good because it prevented illegal peripherals being used in the machine. However, it meant that upgrading a product with a newer one or interchanging equivalent peripherals from different manufacturers became impossible as there were sometimes strong logistical reasons why machine firmware could not be upgraded as well.

Header	Command
245	Request equipment category id
246	Request manufacturer id
244	Request product code
241	Request software revision
197	Request ROM checksum
192	Request build code
004	Request comms revision

Consider a new generation of coin acceptors from the same peripheral manufacturer. The equipment category id would remain as ‘Coin Acceptor’ but the product code, software revision, build code, ROM checksum and comms revision are all likely to change. If the host machine does not verify any of these fields then no problems would be expected. But if they are, and for it to work, the new product would have to emulate, or ‘spoof’, the old product.

Although the old product could be spoofed quite easily, it would be a requirement for the peripheral manufacturer that the actual, TRUE, identity of the peripheral could be read. This is important for field support and traceability reasons. Therefore it is recommended that product spoofing should be tolerated within the confines of the ccTalk specification but that there needs to be some way of recovering the true product identity. This could be through header 255, ‘Factory set-up and test’ since a public command would not be needed for manufacturer-specific support equipment. An internal software switch would allow the true identity of these fields to be revealed.

Discussions are ongoing in this area.

4.6 Cancelled Credit Events

Filing Code	RFC/006
Filing Date	05/04/12
Compatibility Issues	Would require updates to host machine software to support
Current Status	Rejected
Posted	Alex Pogossov, Microsystem Controls Pty Ltd

Following discussions of cancelling credits through a new 'credit type N cancelled' event code…

We do not like the idea of the credits retrospectively cleared by the host either. We also have had problems with manipulation. We also would not issue a credit if coin is to slow, too fast, too early, too late (on a credit sensor), etc., in any case which might suggest manipulation. We would give a respective error code instead, like Money Control does.

However, in some cases, as I mentioned, coins can be swallowed if they are dirty or a cashbox is full. As a result:

- customers may unjustly suffer
- discrepancies are found between the credited money and contents of cashboxes

Kiosk owners want to explain these issues better -- to resolve complaints and also keep an eye on honesty of their own cash collectors.

So I suggest the following:

1. Scenario 1: Cashbox full.
Generate event 'Credit sensor blocked'. No credit is given. Service call.

In addition give error code 'Credit type N cancelled'. If a customer complains, his complaint will be checked against the log. If he claims to have inserted exactly the same coins for which credit cancellation codes are present and around the alleged time, he will be refunded with apologies. If a discrepancy is found after coin counting, this can also be explained.

2. Scenario 2: Coin trapped in gate.
Generate event 'Credit sensor timeout'. No credit is given. Service call.

In addition give error code 'Credit type N cancelled'. If a customer complains, his complaint will be checked against the log. If he claims to have inserted exactly the same coins for which credit cancellation codes are present and around the alleged time, he will be refunded with apologies. If a discrepancy is found after coin counting, this can also be explained.

In conclusion, I think that the addition of a 'Credit type N cancelled' group of informative error codes, similar to the already existing 'Inhibited coin (Type N)', does not contradict Money Controls methodology but only extends/expands upon it and can only be useful, not detrimental, to a coin mech user.

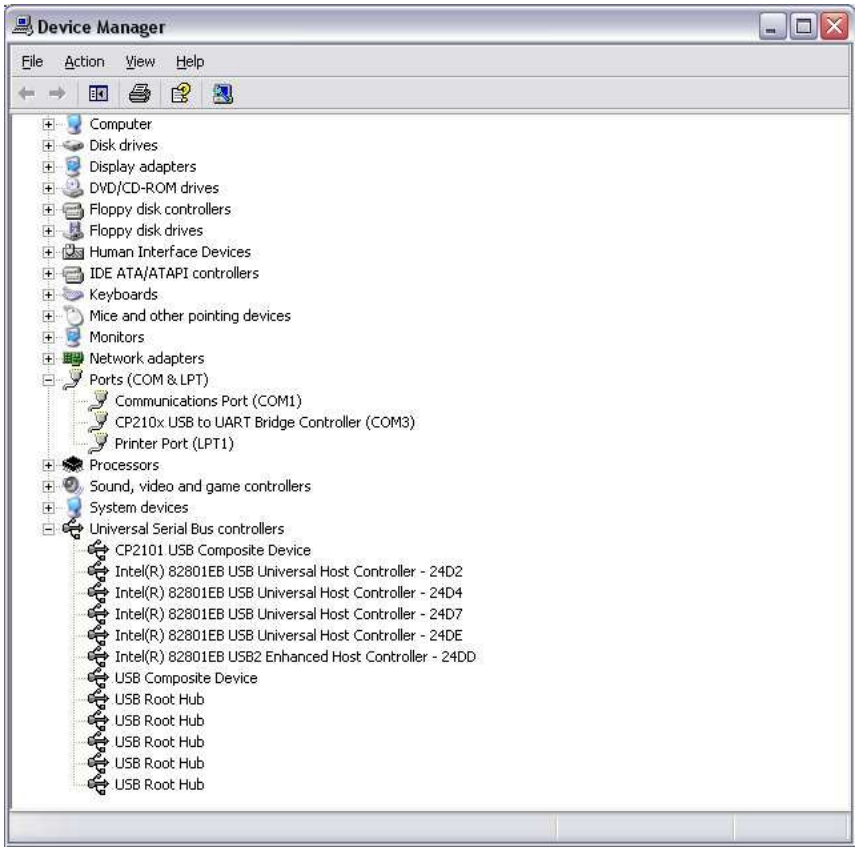
Issuing of such codes or not issuing them can be made configurable, so for backwards compatibility with earlier ccTalk specs these codes can be simply switched off so as not to confuse an ‘old’ host.

Currently, as per our customer request, **as a trial**, we are adding codes ‘Credit type N cancelled’ 160 - 175 (we have 16 categories).

ccTalk over USB

4.7 Introduction

It is possible to run ccTalk over a USB link without changes to the protocol. To do this we can change the physical link layer from 'RS232 at +5V' to USB and retain the packet structure as defined in this specification. If you are familiar with the way USB works then you will be aware that USB provides a set of communication pipes to transfer data, each of which is optimised to a particular type of messaging requirement. The software is written to support different 'classes' of USB device, making use of common functionality where it exists e.g. audio devices, flash memory storage, human input devices etc. Unlike RS232 and UART technology, the complexity of USB requires a PC driver to be loaded during the enumeration of a hot-plugged device. This driver provides the API to talk to the device from the application layer. The use of ccTalk over USB is simplified by having the peripheral enumerate as a COM class device (sometimes called CDC, Communication Device Class). This class emulates a conventional UART link by loading up a virtual COM port into the operating system.



In the example above, a COM3 virtual serial port has been created. Note that the PC does not physically have a serial port 3, but one has been emulated by loading the appropriate USB driver (in this case a CP210x bridge controller).

Once the driver is loaded, it should be possible to set the baud rate, no. of stop bits, odd or even parity etc. just like a normal UART built into the motherboard.

4.8 Advantages

- If your PC or host controller card does not have a UART (as is the case for the newer Intel chip sets) then you have no choice but to run ccTalk through a USB port.
- The 9600 baud rate of ccTalk can be massively increased as full speed USB runs at 12MHz and high speed at 480MHz. It should be possible to achieve a baud rate of at least 1Mbps with low cost devices and no special screening or PCB layout.

4.9 Disadvantages

- USB uses a hub topology. You cannot plug a USB device directly into the USB bus, it must connect to a hub which are themselves connected to the bus. Hubs can be connected to other hubs to increase the number of simultaneous connections. USB provides the addressing mechanism for the host to talk to any peripheral device, and also some data integrity though CRC checksums. Therefore the addressing and checksum fields of ccTalk are redundant when operating over a USB cable. Given the speed increase over USB, a few wasted bytes is not an issue. But if the peripheral side of the link consists of a number of 'virtual' ccTalk devices then the addressing information would still be required.
- A COM class USB device may not be as efficient at transferring large quantities of data as a custom class written specifically for a printer or bill validator for example.

4.10 Hardware Solutions

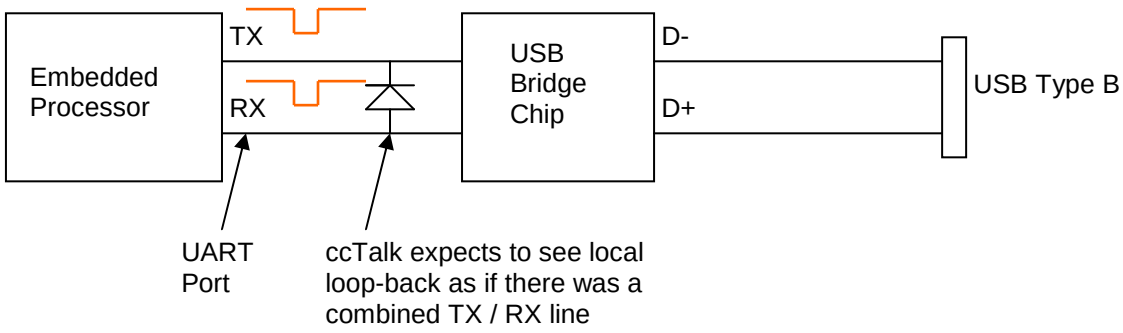
A number of chip manufacturers now supply UART to USB bridge chips which can be embedded into ccTalk devices to make them operate as USB peripherals. On the host side they supply drivers for a range of popular operating systems.

4.11 Example Devices

FTDI FT232RQ USB UART
http://www.ftdichip.com

SiLabs CP2102 USB to UART Bridge
http://www.silabs.com

4.12 System Integration



When the USB lead is plugged into the host PC, the application software in the PC will configure the baud rate in the USB Bridge Chip using the VCOM driver so that the embedded processor can operate as if the USB link was not present.

In this way ccTalk over USB exploits the speed and reliability advantages of USB without any change to the embedded firmware. The protocol is unchanged and any application software is unchanged (perhaps just a change to a different COM port number).

4.13 Broadcast Address and MDCES commands

Since a USB solution connects a single peripheral to a single USB driver, the use of the broadcast address and multi-drop address polling is redundant. The address mechanism is still required for a logical connection but does not act to separate devices on the bus. If in the unlikely event you do not know the ccTalk address of the peripheral then an address poll can be used to return it without the possibility of a clash occurring (only 1 ccTalk peripheral per USB bus).

5. Security Vulnerabilities

5.1 The Null Byte Injection Problem

If somebody gains access to the ccTalk data line and manages to short out the signal to GND for a short time, it will have the effect of injecting a null byte onto the bus i.e. a zero start bit followed by 8 zero data bits. The stop bit will be in the wrong state but not all UARTs are configured to generate a framing error when this occurs.

A null byte sent just prior to a command from the host will force a zero destination address onto the packet.

For example, consider a ‘Modify master inhibit status’ command sent to a coin acceptor.

TX Packet = 002 001 001 228 001 023

Inserting a null byte gives…

TX Packet = 000 002 001 001 228 001 023

So
Destination Address = 0 (the broadcast address, all peripherals respond)
No. of Data Bytes = 2
Source Address = 1
Header = 1 (Reset device)
Data = 228, 1
Checksum = 23 (0 + 2 + 1 + 1 + 228 + 1 + 23 = 256)

The normal reset command is…
TX Packet = 002 000 001 001 252

Adding a zero byte does not interfere with the 8-bit addition checksum so that stays valid.

The effect may be to reset all peripherals attached to the bus, which may or may not lead to a subsequent exploit.

Since the source address becomes the header, most attacks of this sort result in a ‘Reset device’ command since the source address is almost always 1.

There are a number of easy ways to prevent this injection problem on the peripheral.

- Ensure the ‘Reset device’ command takes zero data bytes.
- Trap framing errors if possible and flush any receive data.
- Use CRC checksums if possible because they are more secure.

USB peripherals are not subject to the same kind of ground-shorting attack.

6. Obsolete Commands

The following commands were removed from issue 4.6 of the generic ccTalk specification as they were designed for specific products with limited scope and did not gain any traction in the industry. They have been given the obsolete designation for at least 5 years. The header numbers will be re-used as newer products are developed.

6.1 Header 235 - Read last credit or error code

<<< Obsolete command >>>

Refer to 'Read buffered credit or error codes' for coin acceptors.
Refer to 'Read buffered bill events' for bill validators.

Format (a)
Transmitted data : <none>
Received data : [coin position]

Format (b)
Transmitted data : <none>
Received data : [0] [error code]

Format (c)
Transmitted data : <none>
Received data : [coin position] [sorter path]

6.2 Header 234 - Issue guard code

<<< Obsolete command >>>

Transmitted data : <none>
Received data : ACK

From earlier specifications…
Some commands may be protected by a guard code to improve the reliability in noisy environments. For example, using a command which latches output signals may result in electrical component damage in certain situations. Relying on a single 8-bit checksum to ensure data integrity may not be wise in extremely noisy environments. Therefore, the latch command can be disabled unless it occurs within 50ms (for example) of the guard code command being issued. The slave device will therefore only respond if 2 different commands are received correctly and close together - massively reducing the chances of a random fail.

The reliability of ccTalk in a wide range of actual operating environments is now respected by many manufacturers. There is no need for the guard code command.

6.3 Header 224 - Dispense coins

<<< Obsolete command >>>

This command has been superseded by the ultra-secure 'Dispense hopper coins' command. This obsolete command allows 65,535 coins to be paid out in one go.

Transmitted data : [hopper no.] [no. of coins LSB] [no. of coins MSB]
Received data : ACK or [status code]

[hopper no.]
1, 2, 3 etc.

[status code]
252 - error
253 - not available (incorrect hopper no.)
254 - insufficient coins
255 - busy

This command dispenses a number of coins from the specified change hopper.

A status code is only returned if there is a problem. The usual response is an ACK. Depending on the payout speed, this command may take several minutes to complete. Since the ACK message is returned immediately from the slave device, the 'Request payout status' command should be used to monitor progress.

6.4 Header 223 - Dispense change

<<< Obsolete command >>>

Transmitted data : [coin value LSB] [coin value MSB]
Received data : ACK or [status code]

[coin value]
The actual coin value is a multiple of the 'coin value scaling factor' which may typically be 1, 5 or 10.

[status code]
See 'Dispense coins' for possible status codes.

This command is used to dispense change (coins to a given value) from a changer. It is up to the changer to determine the type and quantity of all coins dispensed. The usual changer algorithm is to dispense the maximum number of highest value coins first, followed by the next highest value etc.

A status code is only returned if there is a problem. The usual response is an ACK. Depending on the payout speed, this command may take several minutes to complete. Since the ACK message is returned immediately from the slave device, the 'Request payout status' command should be used to monitor payout progress.

6.5 Header 220 - One-shot credit

<<< Obsolete command >>>

Format (a)
Transmitted data : [coin position]

Format (b)
Transmitted data : [coin position] [sorter path]

Format (c)
Transmitted data : [0] [error code]

This is a special case where the slave device fires off a credit or error code without being polled by the host. This method is unsuitable for all multi-drop applications (because of collision risk) and does not support a mechanism for re-transmission in the event of a receive error. There is no reply from the host.

6.6 Header 206 - Empty payout

<<< Obsolete command >>>

Format (a)
Transmitted data : <none>
Received data : ACK or [status code]

Format (b)
Transmitted data : [hopper no.]
Received data : ACK or [status code]

[status code]
See 'Dispense coins' for possible status codes.

This command is used to completely empty a coin hopper. The usual response is an ACK - a status code is only returned if there is a problem.

Depending on the payout speed, this command may take several minutes to complete. Since the ACK message is returned immediately from the slave device, the 'Request payout status' command should be used to monitor payout progress.

This command provides an easy way to 'jackpot' a hopper and is best left unimplemented or at the very least PIN number protected.

6.7 Header 205 - Request audit information block

<<< Obsolete command >>>

This command has been tailored to a specific Money Controls product - a changer with 3 hoppers.

Transmitted data : <none>
Received data : [coins in hopper 1A] [coins in hopper 1B]
[coins in hopper 2A] [coins in hopper 2B]
[coins in hopper 3A] [coins in hopper 3B]
[coins to cashbox A] [coin to cashbox B] [coins to cashbox C]
[coins accepted A] [coins accepted B] [coins accepted C]
[coins rejected A] [coins rejected B] [coins rejected C]
[coins paid hop. 1A] [coins paid hop. 1B] [coins paid hop. 1C]
[coins paid hop. 2A] [coins paid hop. 2B] [coins paid hop. 2C]
[coins paid hop. 3A] [coins paid hop. 3B] [coins paid hop. 3C]
[value to cashbox A] [value to cashbox B]
[value to cashbox C] [value to cashbox D]
[coin value scaling factor]

Counter designator A denotes the LSB.

The cashbox value is sent in multiples of the ‘coin value scaling factor’.

6.8 Header 200 - Upload coin data

<<< Obsolete command >>>

Transmitted data : [coin position 1] [coin position 2]
[credit code 1] [credit code 2]
[sorter path 2 | sorter path 1]
[sorter path 4 | sorter path 3]
[op code]
[upload 1] [upload 2] [upload 3] [upload 4]…
Received data : ACK or [reply code]

This command is specific to the remote programming of coins.

[op code]
Bit 0 - credit code control
0 = no action
1 = program
Bit 1 - sorter path control
0 = no action
1 = program
Bit 2 - <undefined>
Bit 3 - <undefined>
Bits 4 to 7 - window programming control
0 = calibrate window

1 = delete window
2 to 15 = <undefined>

[reply code]
<<< Refer to Table 5 >>>

This command allows remote re-programming of coin acceptors by transferring a block of calibration data. The credit code and sorter path may be changed as well.

There is support for dual window programming (useful for repeated banks of coins) and also window deletion.

The upload block itself cannot be larger than 245 bytes for each coin.

6.9 Header 190 - Request payout status

<<< Obsolete command >>>

Format (a)
Transmitted data : <none>
Received data : [events]
[no. of coins paid out LSB] [no. of coins paid out MSB]
[status code]

Format (b)
Transmitted data : <none>
Received data : [events]
[coin value paid out LSB] [coin value paid out MSB]
[status code]

This command is used to monitor the progress of a payout sequence.

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

The event counter is incremented on receipt of a valid hopper payout command and wraps around from 255 to 1. This is a security mechanism to prevent unnecessary retries if a comms error occurs.

[status code]
250 - error, payout blocked (coin over exit sensors)
251 - error, payout timeout (no coins detected)
252 - error, payout jammed (over-current trip)
253 - payout incomplete (insufficient change / incorrect hopper no.)
254 - paying out (busy)
255 - payout completed