Part 2 - Contents

1. Introduction

This document gives a definitive list of all ccTalk commands so far. They are listed in reverse header number order for the following reasons…

- Historical. Headers were defined from the top downwards. The most recent additions to the specification are towards the end.
- Speed. A manual search knowing the header number is quick.

If you are only interested in a specific type of peripheral and are unsure as to which commands need to be implemented then refer to Appendix 1 which is a master command cross-reference. For example, you can find all commands that might be used on a coin acceptor.

There is no requirement on a ccTalk peripheral to implement the full command set. Refer to specific product documentation from each manufacturer to find the level of support.

2. Notation

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

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

Transmitted data in the command description is data sent from the master to the slave.
Received data in the command description is data received by the master from the slave. Note that due to the bi-directional data line, the master may also loop-back its own transmitted data first **- this is not shown**.

For clarity, the full ccTalk message packet is not shown in the command description - **only the data fields**. The destination and source address (if used) will be set accordingly. The length byte will reflect the size of the data packet and the checksum will be calculated for the complete message. The value of the header byte is shown separately for each command.

Values between square brackets, [XXX], indicate message bytes. All data bytes are shown in decimal unless otherwise stated. The bytes on the left are those transmitted or received first.

Some data bytes are shown in binary, e.g. XXXX | 1100 - a total of 8 bits, the lower 4 bits of which are specified.

```
1 indicates a bit set to logic one ( set )
0 indicates a bit set to logic zero ( cleared )
X indicates a bit in a don't care or undefined state
| indicates a field separator ( for display clarity only )
```

Note that **within** a byte, the MSB (most significant bit) is on the left and the LSB (least significant bit) is on the right.

Bulk transfer convention : Data can be **downloaded** to the master from the slave, and **uploaded** to the slave from the master. The slave devices are located 'uphill' in ccTalk parlance.

2.1 Key

<none>	No data bytes. The rest of the message packet is still transmitted or received though.
<variable>	One or more data bytes
ACK	The standard ccTalk ACK message
ASCII	An ASCII string received as characters e.g. 'H' 'e' 'l' 'l' 'o'

3. Command List

3.1 Header 255 - Factory set-up and test

Transmitted data :	<variable>
Received data :	ACK or <variable>

This command is reserved for functions needed during the manufacture of a product and is not intended for general use. Every manufacturer can define their own set of functions buried behind this header number.

3.2 Header 254 - Simple poll

Transmitted data :	<none>
Received data :	ACK

This command can be used to check that the slave device is powered-up and working. No data is returned other than the standard ACK message and no action is performed. It can be used at EMS (Early Morning Start-up) to check that the slave device is communicating. A timeout on this command indicates a faulty or missing device, or an incorrect bus address or baud rate.

***** All ccTalk peripherals must reply to a simple poll** ***

3.3 Header 253 - Address poll

Refer to the section on MDCES.

3.4 Header 252 - Address clash

Refer to the section on MDCES.

3.5 Header 251 - Address change

Refer to the section on MDCES.

3.6 Header 250 - Address random

Refer to the section on MDCES.

3.7 Header 249 - Request polling priority

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

This is an indication by a device of the recommended polling interval for buffered credit information. Polling a device at an interval longer than this may result in lost credits.

[units]
0 - special case, see below
1 - ms
2 - x10 ms
3 - seconds
4 - minutes
5 - hours
6 - days
7 - weeks
8 - months
9 - years

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

3.8 Header 248 - Request status

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

This command reports the status of a coin acceptor.

<<< Refer to Table 4 >>>

3.9 Header 247 - Request variable set

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

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

On a Bill Validator
2 variables have now been universally defined for bill validator products…

[variable 1] - No. of bill types concurrently supported (typically 16 to 64)
[variable 2] - No. of banks supported (typically 1)

3.10 Header 246 - Request manufacturer id

Transmitted data : <none>
Received data : ASCII

The manufacturer's unique identification string is returned.

For example, ‘Money Controls’. Alternatively, it may appear in abbreviated format e.g. ‘MCI’.

<<< Refer to Table 6 >>>

3.11 Header 245 - Request equipment category id

Transmitted data : <none>
Received data : ASCII

The standard equipment category identification string is returned.

<<< Refer to Table 1 >>>

3.12 Header 244 - Request product code

Transmitted data : <none>
Received data : ASCII

The product code is returned. No restriction on format.

The complete product identification string can be determined by using ‘Request product code’ followed by ‘Request build code’.

Caution should be exercised if you are writing a host driver and are checking to see if the product code matches a known string e.g. SCH2. If a newer version of the product is brought out by the peripheral manufacturer e.g. SCH3, then it may mean the peripheral will no longer work with that machine. This may or may not be desirable, depending on the rules governing operation of that machine. The recommendation is to match the string returned by ‘Request equipment category id’ and ‘Request manufacturer id’ rather than ‘Request product code’. If the product code must be used then perhaps only the first few letters are checked e.g. SCHxxx, to cover future upgrades of that product type.

3.13 Header 243 - Request database version

Transmitted data : <none>
Received data : [calibration database no.]

This command retrieves a database number from 1 to 255 which may be used for remote coin programming.

A database number of 0 indicates remote coin programming is not possible.

3.14 Header 242 - Request serial number

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

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

serial 1 = LSB

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

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

3.15 Header 241 - Request software revision

Transmitted data : <none>
Received data : ASCII

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

Any change to slave software, however minor, should be reflected in this revision code.

3.16 Header 240 - Test solenoids

Transmitted data : [bit mask]
Received data : ACK

Implemented on slave devices which use solenoids.

The solenoids are pulsed for a set time. The bit mask indicates which solenoids to operate.

[bit mask]
The following bits have been defined for a coin acceptor :
Bit 0 - Accept gate solenoid. 0 = no action, 1 = pulse.
Bit 1 - Sorter solenoid 1
Bit 2 - Sorter solenoid 2
Bit 3 - Sorter solenoid 3

The slave ACK is usually returned **after** the solenoid de-activates, perhaps 500ms later. For solenoids with opto feedback, the ACK may be returned prior to the solenoid activating to allow header 236, ‘Read opto states’, to report the activation state of the optos.

3.17 Header 239 - Operate motors

Transmitted data : [bit mask]
Received data : ACK

Implemented on slave devices which use motors.

[bit mask]
Each bit controls a motor. 0 = motor off, 1 = motor on.

3.18 Header 238 - Test output lines

Transmitted data : [bit mask]
Received data : ACK

Implemented on slave devices which have an output port.

Various output lines are pulsed. The bit mask indicates which lines to pulse. The length of the pulses is product specific.

[bit mask]
Each bit refers to an output line. 0 = no action, 1 = pulse.

The slave ACK is returned after the line de-activates.

3.19 Header 237 - Read input lines

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

Implemented on slave devices which have an input port.

This command requests various input data from a slave device and is an excellent debugging tool. It can be used to check the operation of switches, push buttons, connector signals, processor input lines etc.

Refer to the product manual for more details.

3.20 Header 236 - Read opto states

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

Implemented on slave devices which have optos.

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

[bit mask]
Each bit refers to an opto. 0 = opto clear, 1 = opto blocked.

3.21 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]

3.22 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.

3.23 Header 233 - Latch output lines

Transmitted data : [bit mask]
Received data : ACK

Implemented on slave devices which have an output port.

Various output lines are latched. The bit mask indicates which lines to latch. Polarities and bit mask interpretation will be detailed in the product manual.

3.24 Header 232 - Perform self-check

Format (a)
Transmitted data : <none>
Received data : [fault code]

Format (b)
Transmitted data : <none>
Received data : [fault code] [extra info]

This command instructs the peripheral device to perform a self-check, i.e. a full diagnostic test without user intervention. The actual level of testing is decided by the slave rather than the host and a fault code is returned. Some slave devices support an additional byte of information for certain fault codes.

Where more than one fault exists on a product, faults will be reported in priority order. Once one fault is fixed, the next fault will be reported. The time to execute this command should be made clear in the product manual.

<<< Refer to Table 3 >>>

3.25 Header 231 - Modify inhibit status

Transmitted data : [inhibit mask 1] [inhibit mask 2]…
Received data : ACK

This command sends an individual inhibit pattern to a coin acceptor or bill validator. With a 2 byte inhibit mask, up to 16 coins or bills can be inhibited or enabled.

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

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

0 = coin / bill disabled (inhibited)
1 = coin / bill enabled (not inhibited)

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

3.26 Header 230 - Request inhibit status

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

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

See ‘Modify inhibit status’ for more details.

3.27 Header 229 - Read buffered credit or error codes

Transmitted data : <none>
Received data : [event counter]
[result 1A] [result 1B]
[result 2A] [result 2B]
[result 3A] [result 3B]
[result 4A] [result 4B]
[result 5A] [result 5B]

This command returns a past history of event codes for a coin acceptor in a small data buffer. This allows a host device to poll a coin acceptor at a rate lower than that of coin insertion and still not miss any credits or other events.

The standard event buffer size is 10 bytes which at 2 bytes per event is enough to store the last 5 events.

A new event ripples data through the return data buffer and the oldest event is lost.

For example, consider a 5 event buffer :

result 5A ⇨ lost
result 5B ⇨ lost

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

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

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

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

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

An event counter is used to indicate any new events and this must be compared at each poll to the last known value.

[event counter] minus [last event counter]	New data stored in result…
0	-
1	1
2	1, 2
3	1, 2, 3
4	1, 2 ,3, 4
5	1, 2, 3, 4, 5
6+	1, 2, 3, 4, 5 & others are lost

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

[result A]
1 to 255 - credit
0 - error code

[result B] for credits
0 - no sorter supported
1 to 8 - sorter path

[result B] for error codes
<<< Refer to Table 2 >>>

The event counter is incremented every time a new credit or error is added to the buffer. When the event counter is at 255 the next event causes the counter to change to 1. The only way for the counter to be 0 is at power-up or reset. This provides a convenient signalling mechanism to the host machine that a major fault has occurred.

Examples

event counter	last event counter	new events
23	23	0
104	102	2
1	255	1
3	253	5
54	48	*one lost event*
0	67	*unknown - power fail*

The result codes are stored in a 2 byte format. If the first result code (result A) is zero then this is a reject or error event rather than a coin credit and the reject / error code is stored in result B. If the first result code is non-zero then the value indicates the type of coin accepted (usually 1 to 12 or 1 to 16) and the second result code (result B) is the sorter route the coin went down. Sorter routes are typically 1 to 4 or 1 to 8 with 0 reserved for no sorter operation.

3.27.1. Event Code at Power-up or Reset

When a peripheral powers-up the event counter will be zero and the event buffer will be zero; 5 pairs of [0] [0] codes. Typically the host software will enable the device for coin acceptance and then nothing will happen until a coin is inserted which may be many hours later. Then the event counter will change to 1 and the host software can read out the new event, which is the first event, from the event buffer. Subsequently, any change in the event counter to zero during polling will indicate that power to the device has been interrupted and the coin acceptor will need re-enabling.

Host software has to be state aware in that it needs to retain a copy of the last event counter so that on reading the current event counter it can calculate how many new events have been generated on the device.

An example of the initialisation logic on the host side is…

```
If events = 0 And lastEvents <> 0 Then ‘unexpected reset !
     lastEvents = 0
     EnableCoins()
ElseIf events = 0 And lastEvents = 0 Then
     primedNewEventFlag = True
ElseIf events <> 0 And lastEvents = 0 And primedNewEventFlag = False Then
     lastEvents = events
End If
```

So if the event counter is zero but the last event counter wasn’t zero then an unexpected reset must have occurred. There is some additional logic to initialise the last event counter with the correct starting value.

There is a potential problem that could occur between the device powering-up and the first event. If the coin acceptor is enabled but a power-loss or reset occurs prior to the first event, the host software will just see a zero event counter and have no knowledge of the device inhibiting itself. When coins are inserted they will be reported as ‘Inhibited coin’.

There are 2 preferred resolutions. One is for the host software to verify or re-apply inhibits as and when necessary. This could be after receiving an unexpected ‘Inhibited coin’ event from the device, or at regular timing intervals. The other is for the device itself to auto-increment to event 1 after being polled with a zero event counter. The device would normally place a null event [0] [0] on the event buffer in this situation (most devices clear RAM during initialisation) and it would be assumed that any host software would simply ignore this null event. However, it is preferable to create a dummy ‘inhibited coin’ event for maximum compatibility with legacy machines. This would be the event code [0] [2].

The product manual for a ccTalk device should make clear which method above is being used and what expectation there is on the host machine software.

3.28 Header 228 - Modify master inhibit status

Transmitted data :	[XXXXXXX | master inhibit status]
Received data :	ACK

[master inhibit status]
Bit 0 only is used.
0 - master inhibit active
1 - normal operation

This command changes the master inhibit status in the slave device. In a coin acceptor, if the master inhibit is active then no coins can be accepted. Likewise for a bill validator.

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

3.29 Header 227 - Request master inhibit status

Transmitted data :	<none>
Received data :	[XXXXXXX | master inhibit status]

This command requests the master inhibit status in the slave device.

Refer to the ‘Modify master inhibit status’ command for polarity.

3.30 Header 226 - Request insertion counter

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

count 1 = LSB

This command returns the total number of coins / bills put through a device.

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

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

3.31 Header 225 - Request accept counter

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

This command returns the total number of coins / bills accepted by a device.

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

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

3.32 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.

3.33 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.

3.34 Header 222 - Modify sorter override status

Transmitted data : [sorter override bit mask]
Received data : ACK

[sorter override bit mask]
B0 - Sorter Path 1
...
B7 - Sorter Path 8

0 = sorter override to a different or default path
1 = no action, normal sorting

This command allows the sorter override status to be set in a coin acceptor. Each bit represents a sorter path for the accepted coin. A zero overrides that sorter path to another one (possibly the *default* sorter path).

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

3.35 Header 221 - Request sorter override status

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

This command returns the sorter override status in a coin acceptor. Each bit represents a sorter path for the accepted coin. A zero means that the sorter path has an active override.

Refer to the 'Modify sorter override status' command for more details.

3.36 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.

3.37 Header 219 - Enter new PIN number

Transmitted data : [PIN 1] [PIN 2] [PIN 3] [PIN 4]
Received data : ACK

Certain commands can be PIN protected - refer to the product manual for a list of commands which support this feature. The existing pin number can be changed with this command, but as this command is itself PIN protected, the existing PIN number must already have been entered.

If any ccTalk command is used with the wrong PIN number (including this command) then there is no response from the slave device.

The PIN number is a 32 bit binary number giving 4,294,967,296 combinations.

Assuming it takes about 50ms per guess, an average of 3.4 years would be needed to crack the code.

A new PIN number of ZERO disables the PIN number mechanism.

3.38 Header 218 - Enter PIN number

Transmitted data :	[PIN 1] [PIN 2] [PIN 3] [PIN 4]
Received data :	ACK

Certain commands can be PIN protected - refer to the product manual for a list of commands which support this function. The existing pin number can be entered with this command. This needs to be done after each power-up or reset.

If the PIN number entered is incorrect then an ACK should still be returned (after a suitable delay) to make exhaustive searching much harder.

3.39 Header 217 - Request payout high / low status

Format (a)

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

Format (b)

Transmitted data :	[hopper no.]
Received data :	[level status]

This command allows the reading of high / low level sensors in a payout system. Multiple hoppers are supported if they exist at the same address.

Bit 0 - Low level sensor status
  0 = higher than or equal to low level trigger
  1 = lower than low level trigger
Bit 1 - High level sensor status
  0 = lower than high level trigger
  1 = higher than or equal to high level trigger
Bit 2 - <reserved>
Bit 3 - <reserved>
Bit 4 - Low level sensor support
  0 = feature not supported or fitted
  1 = feature supported and fitted
Bit 5 - High level sensor support
  0 = feature not supported or fitted
  1 = feature supported and fitted
Bit 6 - <reserved>
Bit 7 - <reserved>

The trigger level is usually fixed mechanically.

3.40 Header 216 - Request data storage availability

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

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

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

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

[read blocks]
1 to 255 - number of blocks of read data available
0 - special case, 256 blocks of read data available

[read bytes]
1 to 252 - number of bytes per block of read data
0 - special case, no read data service

[write blocks]
1 to 255 - number of blocks of write data available
0 - special case, 256 blocks of write data available

[writes bytes]
1 to 251 - number of bytes per block of write data
0 - special case, no write data service

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

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

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

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

3.41 Header 215 - Read data block

Transmitted data : [block number]
Received data : <variable>

[block no.]
0 to 255 (1st block number is always zero)

3.42 Header 214 - Write data block

Transmitted data : [block number] <variable>
Received data : ACK

[block no.]
0 to 255 (1st block number is always zero)

The return ACK from the slave device is only sent after a write cycle has been performed. It us up to the host software whether a verify operation is performed by reading back the data and comparing it.

3.43 Header 213 - Request option flags

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

This command reads option flags (single bit control variables) from a slave device.

On a Coin Acceptor

Bit Position	0	1
0	Credit Code Format : Coin Position	Credit Code Format : CVF
1	-	-
2	-	-
3	-	-
4	-	-
5	-	-
6	-	-
7	-	-

For a discussion on CVF refer to Appendix 3.2

Credit codes returned with the ‘Read buffered credit or error codes’ are usually in ‘coin position’ format. The code does not reflect the monetary value of the coin but the position it is stored within the memory of the coin acceptor. This position usually corresponds directly to the inhibit position. So ‘credit 1’ can be inhibited with bit 0 of the inhibit mask, ‘credit 2’ with bit 1 etc.

On a Bill Validator

Bit Position	Feature (1 = supported)
0	stacker
1	escrow
2	individual bill accept counters
3	individual error counters
4	non-volatile counters
5	bill teach facility
6	bill security tuning
7	remote bill programming

3.44 Header 212 - Request coin position

Transmitted data : [credit code]
Received data : [position mask 1] [position mask 2]

This command can be used in coin acceptors to locate the inhibit position of a given coin based on its ‘credit code’. The inhibit position ties up with the 'Modify inhibit status' command for inhibiting individual coins. To inhibit the coin, the data returned would need to be inverted before sending back out with the ‘Modify inhibit status’ command.

[credit code]
The value returned by the ‘Read buffered credit or error codes’ command.

3.45 Header 211 - Power management control

Transmitted data : [power option]
Received data : ACK

This command can be used to switch slave devices in and out of low power modes if they support power management.

[power option]
0 - normal operation (automatic power switching if supported)
1 - switch to low power mode
2 - switch to full power mode
3 - shutdown

A shutdown normally requires an external reset or a power-down cycle to recover.

3.46 Header 210 - Modify sorter paths

Format (a)
Transmitted data : [coin position] [path]
Received data : ACK

Format (b)
Transmitted data : [coin position] [path 1] [path 2] [path 3] [path 4]
Received data : ACK

This command modifies the sorter path for each coin position in a coin acceptor. Some coin acceptors support multiple sorter paths (override paths) and so a longer version of the command exists. In this case the primary path is the first one specified (path 1).

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

[sorter path]
e.g. 1 to 8

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

3.47 Header 209 - Request sorter paths

Format (a)
Transmitted data : [coin position]
Received data : [path]

Format (b)
Transmitted data : [coin position]
Received data : [path 1] [path 2] [path 3] [path 4]

This command allows sorter paths to be requested in a coin acceptor.

See the ‘Modify sorter paths’ command for more details.

3.48 Header 208 - Modify payout absolute count

Format (a)
Transmitted data : [no. of coins LSB] [no. of coins MSB]
Received data : ACK

Format (b)
Transmitted data : [hopper no.] [no. of coins LSB] [no. of coins MSB]
Received data : ACK

This command allows the hopper coin counter (absolute number of coins in the hopper) to be initialised to a known value. Multiple hoppers are supported if they exist at the same address.

Some hoppers do not support an absolute count value as they can only count coins dispensed.

3.49 Header 207 - Request payout absolute count

Format (a)
Transmitted data : <none>
Received data : [no. of coins LSB] [no. of coins MSB]

Format (b)
Transmitted data : [hopper no.]
Received data : [no. of coins LSB] [no. of coins MSB]

This command allows the hopper coin counter (absolute number of coins in the hopper) to be requested. Multiple hoppers are supported if they exist at the same address.

Some hoppers do not support an absolute count value as they can only count coins dispensed.

3.50 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.

3.51 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'.

3.52 Header 204 - Meter control

Format (a) - set meter to value
Transmitted data : [0] [count 1] [count 2] [count 3]
Received data : ACK

Format(b) - increment meter
Transmitted data : [1]
Received data : ACK

Format (c) - decrement meter
Transmitted data : [2]
Received data : ACK

Format (d) - reset meter
Transmitted data : [3]
Received data : ACK

Format (e) - read meter
Transmitted data : [4]
Received data : [count 1] [count 2] [count 3]

meter value = [count 1] + 256 * [count 2] + 65536 * [count 3]

This meter command is designed for test purposes only (e.g. counting coins on a life-test jig). **It is not secure enough to be used in an auditing environment**.

3.53 Header 203 - Display control

Format (a) - send character (append)
Transmitted data : [0] [char]
Received data : ACK

Format (b) - clear display
Transmitted data : [1]
Received data : ACK

Format (c) - send control code
Transmitted data : [2] [control code]
Received data : ACK

Format (d) - send string (append)
Transmitted data : [3] [char 1] [char 2] [char 3]…
Received data : ACK

Format (e) - request display size
Transmitted data : [4]
Received data : [characters] [lines]

This is a method for controlling a simple LCD display, typically 16 chars x 2 lines or 20 chars x 2 lines.

The control codes will be specific to a type of display but usually provide the ability to…
a) send cursor to home position
b) set cursor attributes on / off / block / underscore
c) scroll left
d) scroll right
e) backspace
f) blink

3.54 Header 202 - Teach mode control

Format (a)
Transmitted data : [position]
Received data : ACK

Format (b)
Transmitted data : [position] [orientation]
Received data : ACK

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

[orientation]
1 to 4 - teaching bills requires the orientation to be known in advance

Puts the device into ‘teach’ mode. Teach is a mechanism whereby new coins or bills can be ‘learnt’ by entering a representative sample. On existing Money Controls products this is referred to as Teach-and-Run®. Once in teach mode the device should be polled with the ‘Request teach status’ command to see what is happening.

3.55 Header 201 - Request teach status

Format (a) - default
Transmitted data : [0]
Received data : [no. of coins / bills entered] [status code]

Format (b) - abort teach operation
Transmitted data : [1]
Received data : [no. of coins / bills entered] [status code]

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

This command is used to monitor the progress of teach mode. Only when a ‘teach completed’ status message is received can the operation be deemed successful.

The actual teach mechanism is under the full control of the slave device. It decides when enough coins or bills have been entered.

3.56 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.

3.57 Header 199 - Configuration to EEPROM

Transmitted data : <none>
Received data : ACK

This command transfers volatile configuration information for a device from RAM into EEPROM. The information for a coin acceptor could possibly include…

a) inhibit information
b) sorter override information
c) sorter paths

Refer to the product manual for more details.

3.58 Header 198 - Counters to EEPROM

Transmitted data : <none>
Received data : ACK

This command transfers volatile counter information from RAM into EEPROM. The information for a coin acceptor could include…

a) coin insertion counter
b) coin accept counter
c) coin reject counter
d) error or alarm counter

Refer to the product manual for more details.

3.59 Header 197 - Calculate ROM checksum

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

Note: The firmware memory type is no longer exclusively ROM; this command can be used for FLASH memory or any other type of reprogrammable memory. The command name has been retained for historical compatibility.

The method of calculating the ROM checksum is not defined in this document and can be adapted to suit the slave device. A simple 'unsigned long' addition can be used as the simplest method with the result displayed as 8 hex digits. The start address and end address is left to the slave device. More powerful devices may choose to calculate a **CRC checksum** using a pre-determined seed value.

There is currently no ccTalk mechanism to compare the returned checksum value with a master reference value. That operation must be done by the host.

3.60 Header 196 - Request creation date

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

The creation date is also known as the…
- manufacturing date
- factory setup date

The date code is stored in a special Money Controls format called RTBY (Relative To Base Year), originally chosen in the mid-Eighties to avoid any Y2K issues.

[date code]

Bit Positions							
15	14	13	9	8	5	4	0
Reserved		Year		Month		Day	
-		Relative		1 to 12		1 to 31	

The year is always stored relative to the PRODUCT_BASE_YEAR which will either be in the product manual or available using the 'Request base year' command. The creation date is a maximum of 31 years after the base year.

3.61 Header 195 - Request last modification date

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

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

The date code formatted is detailed under the 'Request creation date' command.

3.62 Header 194 - Request reject counter

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

This command returns the total number of reject coins / bills put through a device.

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

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

3.63 Header 193 - Request fraud counter

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

This command returns the total number of fraud coins / bills put through a device.

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

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

3.64 Header 192 - Request build code

Transmitted data : <none>
Received data : ASCII

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

3.65 Header 191 - Keypad control

Transmitted data : [keypad code]
Received data : [cursor position] [char 1] [char 2] [char 3]…

This command allows the remote execution of a keypad and display control system. The code sent is used to simulate which button was pressed on the keypad and the message sent back is the display string together with a current cursor position.

For displays that support more than one line a null byte can be used in the string to indicate line breaks.

[keypad code]
0 - return current display message
1 - press button 1
2 - press button 2
3 - press button 3
4 - press button 4
etc.

[cursor position]
0 - cursor off or not used
1 to 255 - current cursor position

3.66 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

3.67 Header 189 - Modify default sorter path

Transmitted data : [default path]
Received data : ACK

[default path]
e.g. 1 to 4, 1 to 8

This command allows the default sorter path on a coin acceptor to be changed. If there is an active override on the current coin sorter path then it will be routed to the default path.

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

3.68 Header 188 - Request default sorter path

Transmitted data : <none>
Received data : [default path]

This command reads the default sorter path on a coin acceptor.

See the 'Modify default sorter path' command for more details.

3.69 Header 187 - Modify payout capacity

Format (a)
Transmitted data : [no. of coins LSB] [no. of coins MSB]
Received data : ACK

Format (b)
Transmitted data : [hopper no.] [no. of coins LSB] [no. of coins MSB]
Received data : ACK

This command sets the maximum number of coins a payout device can hold. Multiple hoppers are supported if they exist at the same address.

3.70 Header 186 - Request payout capacity

Format (a)
Transmitted data : <none>
Received data : [no. of coins LSB] [no. of coins MSB]

Format (b)
Transmitted data : [hopper no.]
Received data : [no. of coins LSB] [no. of coins MSB]

This command requests the maximum number of coins a payout device can hold. Multiple hoppers are supported if they exist at the same address.

3.71 Header 185 - Modify coin id

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

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

Some coin acceptors can store an identification string alongside the normal validation parameters for each coin. Refer to the product manual for details.

Refer to Appendix 3.1 for coin naming information.

3.72 Header 184 - Request coin id

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

Refer to the 'Modify coin id' command for more details.

3.73 Header 183 - Upload window data

Format (a) - program coin
Transmitted data : [0] [coin position]
<variable>
Received data : ACK

Format (b) - modify credit code
Transmitted data : [1] [coin position] [credit code]
Received data : ACK

Format (c) - delete coin
Transmitted data : [2] [coin position]
Received data : ACK

Format (d) - program token
Transmitted data : [3] [token position]
<variable>
Received data : ACK

Format (e) - delete token
Transmitted data : [4] [token position]
Received data : ACK

This command is used for the remote programming of coins or tokens.

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

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

The variable block of data sent to program a coin or token will be manufacturer specific.

Currently there is no mechanism provided for changing the credit codes of tokens as tokens are usually substituted into coin window space and inherit their attributes.

Some or all of the above features may be disabled for security reasons.

3.74 Header 182 - Download calibration info

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

This command is used to support the remote coin programming operation. No further details are given here.

3.75 Header 181 - Modify security setting

Transmitted data : [coin / bill position] [security setting]
Received data : ACK

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

[security setting]

0	- default setting (nominal performance)
1 to 7	- gradually increase fraud rejection (7 steps)
255 to 249	- gradually increase true acceptance (7 steps)
8 to 248	- undefined

This command allows a validator's performance to be finely tuned. On all known devices there is a trade-off between fraud rejection and true acceptance.

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

3.76 Header 180 - Request security setting

Transmitted data : [coin / bill position]
Received data : [security setting]

Refer to the 'Modify security setting' command for more details.

3.77 Header 179 - Modify bank select

Transmitted data : [bank no.]
Received data : ACK

[bank no.]
0 - default bank
1 to 255 - alternative banks

Some devices can support multiple banks of coins / bills subject to the condition of only a single bank being enabled (active) at any one time. This command allows the host controller to switch between banks. Assuming a typical device operates with 16 coins or bills, this command expands the potential capability to 16 x 256 = 4,096 coins or bills, though not all can be accepted concurrently.

Coins or bills within a bank can be controlled with the ‘Modify inhibit status’ command.

3.78 Header 178 - Request bank select

Transmitted data : <none>
Received data : [bank no.]

Refer to the ‘Modify bank select’ command for details.

3.79 Header 177 - Handheld function

Transmitted data : [XX | mode | function] <variable>
Received data : <variable>

[mode]
0 to 3 - operating mode

[function]
0 to 15 - operating function

This command may be used to support simple handheld toolkits (non-PC based, low processing power) which send simple ‘mode / function’ sequence commands to the slave device. The response may be an ACK or a number of data bytes.

This command will be product and manufacturer specific and may not be implemented at all. Some devices have useful debugging features which can be accessed through this command.

See ‘BACTA Token Selection’ in Part 3 of the documentation for an example of use.

3.80 Header 176 - Request alarm counter

Transmitted data : <none>
Received data : [alarm count]

This command returns the number of alarm events since the last request was sent (i.e. it is cleared on reading). It is only used on products which support an alarm output line.

3.81 Header 175 - Modify payout float

Format (a)
Transmitted data : [no. of coins LSB] [no. of coins MSB]
Received data : ACK

Format (b)
Transmitted data : [hopper no.] [no. of coins LSB] [no. of coins MSB]
Received data : ACK

This command sets the working ‘float’ level for a payout device. Multiple hoppers are supported if they exist at the same address.

3.82 Header 174 - Request payout float

Format (a)
Transmitted data : <none>
Received data : [no. of coins LSB] [no. of coins MSB]

Format (b)
Transmitted data : [hopper no.]
Received data : [no. of coins LSB] [no. of coins MSB]

This command requests the working ‘float’ level for a payout device. Multiple hoppers are supported if they exist at the same address.

3.83 Header 173 - Request thermistor reading

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

Previously : Some products use a thermistor to provide an approximate method of determining the ambient temperature. This command returns the raw thermistor reading as a byte.

This command is no longer restricted to thermistor temperature measurement – more accurate methods are commonly available. The format of the return data is now specified as ‘degrees Celsius’ in 2’s complement format. This gives a range of -128°C to +127°C.

Some Money Controls products return thermistor data in a ‘raw’ format as described below.

The thermistor is placed in a simple voltage divider network biased such that the value of 128 indicates a temperature of 25°C. A value higher than 128 means a temperature above 25°C and a value lower than 128 means a temperature below 25°C. The thermistor circuit consists of an analogue input voltage to the microcontroller, with a 10K pull-down to GND and a 10K NTC thermistor pull-up to Vref. As the temperature goes up, so does the input voltage. The signal is reasonably linear around 25°C but curves at the extremes. This formula can be used to calculate the approximate temperature from the thermistor reading.

```
temperature = (((thermistor value - 128) / 102 * 45) + 25) °C
```

3.84 Header 172 - Emergency stop

Transmitted data : <none>
Received data : [payout coins remaining]

This command immediately halts the payout sequence and reports back the number of coins which failed to be paid out.

3.85 Header 171 - Request hopper coin

Transmitted data : <none>
Received data : ASCII

This command returns the name of the coin that the hopper can pay out.

Refer to Appendix 3.1 for coin naming information.

3.86 Header 170 - Request base year

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

The product base year (see PRODUCT_BASE_YEAR in ‘Request creation date’ and ‘Request last modification date’ commands) is returned as a 4 character ASCII string.

e.g. ‘1999’, ‘2000’.

3.87 Header 169 - Request address mode

Transmitted data : <none>
Received data : [address mode]

This command returns the ccTalk addressing mode to help with automatic re-configuration of ccTalk peripherals. Its use is informational only.

[address mode]
Bit mask :
B0 - Address is stored in Flash / ROM
B1 - Address is stored in RAM
B2 - Address is stored in EEPROM or equivalent
B3 - Address selection via interface connector
B4 - Address selection via PCB links
B5 - Address selection via switch
B6 - Address may be changed with serial commands (volatile)
B7 - Address may be changed with serial commands (non-volatile)

Any bits may be set according to the operating mode of the device.
[0] is returned to indicate another type of addressing mode.

3.88 Header 168 - Request hopper dispense count

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

The dispense counter records the number of coins dispensed by the hopper.

count 1 = LSB, count 3 = MSB

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

3.89 Header 167 - Dispense hopper coins

Format (a) - Money Controls ‘SCH2’ version
Transmitted data : <variable> [no. of coins]
Received data : [event counter] or NAK

Format (b) - Money Controls ‘SCH1’ version
Transmitted data : <variable> [no. of coins]
Received data : ACK or NAK

This command is used to securely dispense between 1 and 255 coins from a hopper. The variable data represents special security codes which have to be correct for a coin payout to be successful. The security algorithm is not given in this document.

The event counter returned is the post-increment value. Refer to the ‘Request hopper status’ command for more details.

3.90 Header 166 - Request hopper status

Transmitted data : <none>
Received data : [event counter] [payout coins remaining]
[last payout : coins paid] [last payout : coins unpaid]

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

The event counter is incremented every time a valid dispense command is received. When the event counter is at 255 the next event causes the counter to change to 1. The only way for the counter to be 0 is at power-up or reset.

The [payout coins remaining], [last payout : coins paid] and [last payout : coins unpaid] fields are updated appropriately during and after coin payout.

Refer to the Money Controls SCH2 hopper manual for more details.

3.91 Header 165 - Modify variable set

Transmitted data : <variable>
Received data : ACK

This command modifies variable data on the slave device. Refer to the product manual for more details.

3.92 Header 164 - Enable hopper

Transmitted data : [enable code]
Received data : ACK

[enable code]
165 - enable hopper payout
not 165 - disable hopper payout

This command must be used to enable a hopper before paying out coins.

The value 165 is 'A5' in hex and '10100101' in binary.

3.93 Header 163 - Test hopper

Format (c) - Money Controls 'SCH3' version
Transmitted data : <none>
Received data : [hopper status register 1] [hopper status register 2]
[hopper status register 3]

Format (b) - Money Controls ‘SCH2’ version
Transmitted data : <none>
Received data : [hopper status register 1] [hopper status register 2]

Format (a) - Money Controls ‘SCH1’ version
Transmitted data : <none>
Received data : [hopper status register 1]

This command reports back various operating and error flags from the hopper and is the equivalent of the ‘Perform self-check’ command in coin / bill acceptors.

[hopper status register 1]
Bit mask :
Bit 0 - Absolute maximum current exceeded
Bit 1 - Payout timeout occurred
Bit 2 - Motor reversed during last payout to clear a jam
Bit 3 - Opto fraud attempt, path blocked during idle
Bit 4 - Opto fraud attempt, short-circuit during idle
Bit 5 - Opto blocked permanently during payout
Bit 6 - Power-up detected
Bit 7 - Payout disabled

[hopper status register 2]
Bit mask :
Bit 0 - Opto fraud attempt, short-circuit during payout
Bit 1 - Single coin payout mode
Bit 2 - Use other hopper for change
Bit 3 - Opto fraud attempt, finger / slider mis-match
Bit 4 - Motor reverse limit reached
Bit 5 - Inductive coil fault
Bit 6 - NV memory checksum error
Bit 7 - PIN number mechanism

[hopper status register 3]
Bit mask :
Bit 0 - Power-down during payout
Bit 1 - Unknown coin type paid
Bit 2 - PIN number incorrect
Bit 3 - Incorrect cipher key
Bit 4 – Encryption enabled
Bit 5 - < Reserved >
Bit 6 - < Reserved >
Bit 7 - < Reserved >

1 = condition on, 0 = condition off

3.93.1. Explanation

3.93.1.1. Absolute maximum current exceeded

Payout stopped because a maximum threshold current was exceeded. This is the software equivalent of a protection fuse and may indicate an irrecoverable jam or a motor bridge fault. The flag must be cleared with a software reset prior to next payout but if the fault is permanent then it may not be possible to clear this condition.

3.93.1.2. Payout timeout occurred

Indicates the payout operation terminated after a specified timeout period with no coins visible on the exit sensor. This could be because the hopper was empty. Warning flag only.

3.93.1.3. Motor reversed during last payout to clear a jam

A motor jam was detected and the hopper attempted a reverse to clear it. Warning flag only.

3.93.1.4. Opto fraud attempt, path blocked during idle

The hopper exit opto saw a blockage outside a payout operation. This could be due to a coin stuck in the exit path or a deliberate fraud attempt. The flag must be cleared with a software reset prior to next payout.

3.93.1.5. Opto fraud attempt, short-circuit during idle

The hopper exit opto saw a short-circuit outside a payout operation. This is most likely due to shining light directly into the exit opto. The flag must be cleared with a software reset prior to next payout.

3.93.1.6. Opto blocked permanently during payout

The hopper exit opto saw a blockage for too long a time while paying out coins. This could be due to a coin stuck in the exit path or a deliberate fraud attempt. The flag must be cleared with a software reset prior to next payout.

3.93.1.7. Power-up detected

This flag indicates that the hopper has had power applied to it. It is cleared by a software reset and so can be used to distinguish between a software reset and a hardware reset / power supply removal. Warning flag only.

3.93.1.8. Payout disabled

The hopper is always disabled after a power-up or reset. The flag must be cleared with an ‘Enable hopper’ command prior to next payout.

3.93.1.9. Opto fraud attempt, short-circuit during payout

The hopper exit opto saw a short-circuit while paying out coins. This is most likely due to shining light directly into the exit opto. The flag must be cleared with a software reset prior to next payout.

3.93.1.10. Single coin mode

This flag indicates the hopper is working in single coin mode and will only respond to commands to dispense one coin at a time. This method is slower but achieves maximum security as there is full handshaking on each coin paid out. Status flag only.

3.93.1.11. Use other hopper for change

Accumulator mode : This flag indicates the hopper cannot dispense any more coins and a secondary hopper must be used to complete the dispense operation. Status flag only.

3.93.1.12. Opto fraud attempt, finger / slider mis-match

Some hoppers are fitted with sensors on the fingers or sliders used to eject coins cleanly. If a coin is seen leaving the exit optos but no signal is seen on the finger or slider then this error flag is set as it could be a possible fraud attempt. The flag must be cleared with a software reset prior to next payout.

3.93.1.13. Motor reverse limit reached

If the number of sequential motor reverses exceeds a preset limit then the hopper dispense is aborted and this flag set. It is likely that there is a permanent coin jam in the hopper. The flag must be cleared with a software reset prior to next payout.

3.93.1.14. Inductive coil fault

Accumulator mode : This flag indicates a fault has been detected with the coil used to distinguish between different coin types. The flag must be cleared with a software reset prior to next payout but if the fault is permanent then it may not be possible to clear this condition.

3.93.1.15. NV memory checksum error

An error has occurred with the NV memory which means the values of the coin paid and unpaid counters may not be correct. Refer to the product manual for appropriate action.

3.93.1.16. PIN number mechanism

This flag indicates that the PIN number mechanism is enabled and that a PIN has to be entered using command header 218 before any coins can be paid out. Status flag only.

3.93.1.17. Power-down during payout

If power is lost during a payout operation then this flag will be set. The values of the paid and unpaid counters will indicate what happened just prior to power being lost. The flag must be cleared with a software reset prior to next payout.

3.93.1.18. Unknown coin type paid

Accumulator mode : This flag indicates that the coin just paid was not a recognised type. The hopper stops immediately. Either an illegal coin type has been placed in the hopper or there is a problem with the coin sensing. The flag must be cleared with a software reset prior to next payout.

3.93.1.19. PIN number incorrect

The hopper dispense operation failed because the PIN number protection mechanism has been enabled (using the ‘Enter new PIN number’ command) - see PIN number mechanism flag - but the ‘Enter PIN number’ command has not yet been sent or the number sent was incorrect. The flag must be cleared with a software reset prior to next payout.

3.93.1.20. Incorrect cipher key

If the hopper uses encryption and an incorrect cipher key is sent then the hopper dispense operation will fail and this flag set. The flag must be cleared with a software reset prior to next payout.

3.93.1.21. Encryption enabled

If the hopper requires a cipher key to be calculated when dispensing coins then this flag is set. Status flag only.

Header 162 - Modify inhibit and override registers

Transmitted data :	[current : inhibit mask 1] [current : inhibit mask 2]
	[current : sorter override bit mask]
	[next : inhibit mask 1] [next : inhibit mask 2]
	[next : sorter override bit mask]
Received data :	ACK

For coin acceptors this command…
a) sets inhibits and overrides in one operation (see also the ‘Modify inhibit status’ command and the ‘Modify sorter override status’ command)
b) sets both a ‘current’ value and a ‘next coin’ value

The benefit of using this command is that it allows precise coin-by-coin control of the coin acceptor and provides a means of tackling the inherent latency in serial operation. If the next coin is always disabled, the validator will only accept one coin at a time. After each credit (strictly speaking an attempted accept sequence) the host software can request another coin and thus control the overall accept rate. If the next coin overrides are always active, the validator will only route one coin at a time to a payout tube - all other coins will go to the cashbox. The host software can thus maintain a precise fill level in any tube.

3.94 Header 161 - Pump RNG

Transmitted data :	<variable>
Received data :	ACK

This command ‘pumps’ the random number generator of the slave device with a set of random numbers and is part of the hopper dispense encryption algorithm.

No further details are given here.

3.95 Header 160 - Request cipher key

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

This command requests a cipher key from the slave device and is part of the hopper dispense encryption algorithm. No further details are given here.

3.96 Header 159 - Read buffered bill events

Transmitted data : <none>
Received data : [event counter]
[result 1A] [result 1B]
[result 2A] [result 2B]
[result 3A] [result 3B]
[result 4A] [result 4B]
[result 5A] [result 5B]

This command returns a history of bill events in a similar way to that of a coin acceptor. The most recent event is in result 1, the oldest in result 5. A total of 5 events can be reported with each event stored in 2 bytes. Refer to the 'Read buffered credit or error codes' command for more explanation.

For an explanation of the bill event codes…
<<< Refer to Table 7 >>>

Note that a credit code of 255 is reserved for a coupon or ticket. For an explanation of how this works, refer to Appendix 16 - Bill Acceptor Messaging Example.

3.97 Header 158 - Modify bill id

Transmitted data : [bill type] [char 1] [char 2] [char 3]…
Received data : ACK

[bill type]
e.g. 1 to 16

Some bill validators can store an identification string alongside the normal validation parameters for each bill. Refer to the product manual for details.

A 7 character identification code is used…

[C] [C] [V] [V] [V] [V] [I]

CC	= Standard 2 letter country code e.g. GB for the U.K. (Great Britain)
VVVV	= Bill value in terms of the country scaling factor
I	= Issue code. Starts at A and progresses B, C, D, E…

See Appendix 10 for country codes and Appendix 15 for more information on this command.

3.98 Header 157 - Request bill id

Transmitted data : [bill type]
Received data : [char 1] [char 2] [char 3]…

Refer to the 'Modify bill id' command for more details.

3.99 Header 156 - Request country scaling factor

Transmitted data : [country char 1] [country char 2]
Received data : [scaling factor LSB] [scaling factor MSB] [decimal places]

This command requests the scaling factor and decimal places for the standard country code provided.

If all the return bytes are zero then that country code is not supported.

3.100 Header 155 - Request bill position

Transmitted data : [country char 1] [country char 2]
Received data : [position mask 1] [position mask 2]…

This command can be used in bill acceptors to locate the inhibit mask of a given currency based on its country code. The inhibit mask ties up with the 'Modify inhibit status' command for inhibiting individual bills.

If this data is used directly as the new inhibit mask then the currency selected will be enabled and all other currencies inhibited.

3.101 Header 154 - Route bill

Transmitted data : [route code]
Received data : ACK or [error code]

This command controls routing of a bill held in an escrow.

[route code]
0 - return bill
1 - send bill to cashbox / stacker
255 - extend escrow timeout

[error code]
254 - escrow is empty
255 - failed to route bill

Route code 255 resets the default escrow timeout to prevent the note being returned to the customer. Although the escrow timeout is usually quite long (30s or more) there may be a situation where the host machine needs more time before a routing decision is made.

3.102 Header 153 - Modify bill operating mode

Transmitted data : [mode control mask]
Received data : ACK

This command controls whether various product features are used.

[mode control mask]
B0 - stacker
B1 - escrow

0 = do not use, 1 = use

3.103 Header 152 - Request bill operating mode

Transmitted data : <none>
Received data : [mode control mask]

Refer to the 'Modify bill operating mode' command.

3.104 Header 151 - Test lamps

Transmitted data : [lamp no.] [lamp control]
Received data : ACK

This command can be used to control lamps on products that have them. On bill validators, front-panel lamps are often used to guide the user on note insertion. Other products may have status LEDs etc.

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

[lamp control]
0 - automatic mode, allow device to control lamp
1 - manual mode, force lamp off
2 - manual mode, force lamp on
10 to 255 – manual mode, force lamp to flash (50% duty cycle, on time in 10ms steps)

In manual flash mode…
Lamp control = 10 = 200ms period = 5 flashes per second
Lamp control = 17 = 340ms period = 3 flashes per second approx.
Lamp control = 25 = 500ms period = 2 flashes per second
Lamp control = 50 = 1000ms period = 1 flash per second
Lamp control = 100 = 2000ms period = 1 flash every 2 seconds
Lamp control = 250 = 5000ms period = 1 flash every 5 seconds

3.105 Header 150 - Request individual accept counter

Transmitted data : [bill type]
Received data : [count 1] [count 2] [count 3]

Some bill validators support the individual counting of different bill types. This command returns the number of bills accepted of a given type. The type is that reported by the 'Read buffered bill events' command.

count 1 = LSB

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

3.106 Header 149 - Request individual error counter

Transmitted data : [error type]
Received data : [count 1] [count 2] [count 3]

Some bill validators support the individual counting of different error types. This command returns the number of errors recorded of a given type.

[error type]
Refer to Product Manual or Table 7.

count 1 = LSB

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

3.107 Header 148 - Read opto voltages

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

This command returns a series of scaled voltages for a device using optos (e.g. a bill validator) rather than the blocked / clear indication which would be returned by the 'Read opto states' command.

Refer to the product manual for the format of this data which may typically be 8 bits or 16 bits and scaled to 3.3V or 5.0V.

3.108 Header 147 - Perform stacker cycle

Transmitted data : <none>
Received data : ACK or [error code]

[error code]
254 - stacker fault
255 - stacker not fitted

This command executes 1 cycle of the stacker on a bill validator for diagnostic purposes. The ACK is returned after the cycle completes.

3.109 Header 146 - Operate bi-directional motors

Transmitted data : [motor mask] [direction flags] [speed]
Received data : ACK

This command is a diagnostic tool for testing PWM controlled motors.

[motor mask]
Each bit represents a motor. 0 = off, 1 = on.

[direction flags]
Each bit represents a motor direction. 0 = backwards, 1 = forwards.

[speed]
0 - default speed
1 to 255 - relative PWM speed. 1 = slowest, 255 = fastest.

The data format allows any combination of motors to be turned on and off in a single command, either in a forwards or backwards direction, but with a shared PWM speed. It is not possible to assign different speeds to different motors.

As an example, consider motor A controlled by bit 0 and motor B controlled by bit 1.

```
[ 1 ] [ 1 ] [ 128 ]   motor A forwards at speed 128
[ 2 ] [ 2 ] [ 128 ]   motor B forwards at speed 128
[ 1 ] [ 0 ] [ 128 ]   motor A backwards at speed 128
[ 2 ] [ 0 ] [ 128 ]   motor B backwards at speed 128
[ 3 ] [ 1 ] [ 255 ]   motor A forwards at speed 255
                      motor B backwards at speed 255
[ 3 ] [ 2 ] [ 255 ]   motor A backwards at speed 255
                      motor B forwards at speed 255
[ 3 ] [ 3 ] [ 0 ]     motor A forwards at default speed
                      motor B forwards at default speed
[ 3 ] [ 0 ] [ 0 ]     motor A backwards at default speed
                      motor B backwards at default speed
[ 0 ] [ 0 ] [ 0 ]     motor A stopped, motor B stopped
```

3.109.1. Alternative Format

The alternative format interpretation allows different motors to have different PWM speeds which may be advantageous on some products.

There are 2 changes to the above specification…
(a) The [motor mask] is used to **select** a motor rather than to turn it on and off.
(b) The [speed] value of zero, the default speed, is defined as **stopped**.

As an example, consider motor A controlled by bit 0 and motor B controlled by bit 1.

```
[ 1 ] [ 1 ] [ 128 ]   motor A forwards at speed 128
[ 2 ] [ 2 ] [ 128 ]   motor B forwards at speed 128
[ 1 ] [ 0 ] [ 128 ]   motor A backwards at speed 128
[ 2 ] [ 0 ] [ 128 ]   motor B backwards at speed 128
[ 3 ] [ 1 ] [ 255 ]   motor A forwards at speed 255
                      motor B backwards at speed 255
[ 3 ] [ 2 ] [ 255 ]   motor A backwards at speed 255
                      motor B forwards at speed 255
[ 1 ] [ 0 ] [ 0 ]     motor A stopped
[ 2 ] [ 0 ] [ 0 ]     motor B stopped
[ 3 ] [ 0 ] [ 0 ]     motor A stopped, motor B stopped
```

In this format motors can be left running with different PWM speeds.

Most importantly, with the previous format, a motor mask of zero will stop all motors. In the alternative format it will do nothing as no motors are selected.

If a product can only operate a single motor at a time then only the first bit in the motor bit mask, starting at bit 0, will be used.

3.110 Header 145 - Request currency revision

Format (a)
Transmitted data : <none>
Received data : ASCII

Format (b)
Transmitted data : [country char 1] [country char 2]
Received data : ASCII

If no parameters are sent then a general currency revision is returned. Otherwise, 2 ASCII characters identifying the country of interest can be sent to determine a revision code specific to that country.

If the country identifier is not recognised then the string ‘Unknown’ is returned.

3.111 Header 144 - Upload bill tables

Transmitted data : [block] [line] [data 1] [data 2]… [data 128]
Received data : ACK

This commands sends new bill table information into a validator in a manufacturer-neutral format. The data is split into blocks and lines. There are 128 bytes per line, 256 lines per block, up to a maximum of 256 blocks. This gives a total capacity of 256 x 256 x 128 = 8 Mbytes.

Lines shorter than 128 bytes can be sent since each message packet contains a length descriptor.

There is no reference to bill type in the data structure - it is assumed that this is represented internally.

3.112 Header 143 - Begin bill table upgrade

Transmitted data : <none>
Received data : ACK

This command initiates a bill table upgrade.

3.113 Header 142 - Finish bill table upgrade

Transmitted data : <none>
Received data : ACK or NAK

This command terminates a bill table upgrade.

3.114 Header 141 - Request firmware upgrade capability

Transmitted data : <none> or [module identifier]
Received data : [firmware options]

[module identifier]
Where a peripheral consists of a number of sub-peripherals or separate firmware modules on the same ccTalk address, a module identifier can be specified.

[firmware options]
0 - firmware in ROM / EPROM
1 - firmware in FLASH / EEPROM with upgrade capability

Only firmware of type 1 can be upgraded remotely.

3.115 Header 140 - Upload firmware

Transmitted data : [block] [line] [data 1] [data 2]… [data 128]
Received data : ACK

This general purpose command can be used to upgrade the firmware in a validator. The format is the same as the 'Upload bill tables' command.

3.116 Header 139 - Begin firmware upgrade

Transmitted data : <none> or [module identifier]
Received data : ACK

[module identifier]
Where a peripheral consists of a number of sub-peripherals or separate firmware modules on the same ccTalk address, a module identifier can be specified. Headers 140 and 138 which subsequently follow are routed internally to this module.

This command initiates a firmware upgrade.

3.117 Header 138 - Finish firmware upgrade

Transmitted data : <none>
Received data : ACK or NAK

This command terminates a firmware upgrade.

3.118 Header 137 - Switch encryption code

Transmitted data : [sec 2 | sec 1] [sec 4 | sec 3] [sec 6 | sec 5]
Received data : ACK

[sec N]
BCD security digit in the range 0 to 9

The full encryption code is 6 digits long, e.g. 123456, TX = 0x21 0x43 0x65

When the ccTalk encryption layer is being used, this command allows the encryption code or key to be switched to a new value. Regular switching of the encryption code massively increases the security of the communications link.

The new security code takes effect **after** the ACK is returned.

Encryption code changes are volatile. The 'Store encryption code' command can be used to make the change permanent.

3.119 Header 136 - Store encryption code

Transmitted data : <none>
Received data : ACK

This command stores the current encryption code in NV memory. At the next power up, this is the encryption code which needs to be used.

3.120 Header 135 - Set accept limit

Transmitted data : [no. of coins]
Received data : ACK

Some applications such as gaming machines require no more than say 3 or 5 coins to be accepted per game play. For a serial coin acceptor this means sending a command to inhibit coin acceptance after a specified number of coins has been entered. When coin throughput is high there is a chance further coins will be accepted due to the latency of serial itself. New coins can only be detected at the credit polling interval and by that time more coins could have passed the accept gate.

Refer to header 231, ‘Modify inhibit status’ and header 162, ‘Modify inhibit and override registers’. The latter was introduced for coin by coin control but it is not suitable for fast acceptance of a stream of coins.

Set accept limit has a parameter ‘no. of coins’ which determines how many coins are accepted before the coin acceptor inhibits itself. After self-inhibit, no further coins can be accepted until another ‘Set accept limit’ command is sent. During self-inhibit, ccTalk event codes ‘Inhibited coin’ will be returned for each coin entered. It is possible to switch off the self-inhibit mechanism by sending ‘no. of coins’ equal to zero.

Self-inhibit operates independently of the modify inhibit status register used in ccTalk command 231. All coins are inhibited, regardless of type. Therefore this command is really only of use in single-coin coin acceptors. Currently there are no plans for a ‘Set accept value’ command which works as a device-side totaliser.

3.121 Header 134 - Dispense hopper value

Transmitted data : [sec 1] [sec 2] [sec 3] [sec 4]
[sec 5] [sec 6] [sec 7] [sec 8]
[coin value LSB] [coin value MSB]
Received data : [event counter] or NAK

[coin value]
Range 0 to 65,535.

This command is used to securely dispense a coin value from an accumulator hopper. For a discussion of an ‘accumulator hopper’ versus a ‘discriminator hopper’ refer to the glossary. The coin value is based on the lowest unit in that currency, e.g. pence or cents.

The variable data represents special security codes which have to be correct for a coin payout to be successful. The security algorithm is not given in this document. An ‘unencrypted’ hopper ignores the security codes and always dispenses coins.

The event counter returned is the post-increment value. Refer to the ‘Request hopper polling value’ command for more details.

3.122 Header 133 - Request hopper polling value

Transmitted data : <none>
Received data : [event counter]
[payout value remaining LSB] [payout value remaining MSB]
[last payout : coin value paid LSB] [last payout : coin value paid MSB]
[last payout : coin value unpaid LSB] [last payout : coin value unpaid MSB]

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

The event counter is incremented every time a valid dispense command is received. When the event counter is at 255 the next event causes the counter to change to 1. The only way for the counter to be 0 is at power-up or reset.

The [payout value remaining], [last payout : coin value paid] and [last payout : coin value unpaid] fields are updated appropriately during and after coin payout.

When [payout value remaining] = 0, which means both the LSB and MSB bytes are zero, the payout operation has stopped and the paid / unpaid values show whether it was successful. The flags status register (see the Test hopper command) can be checked to determine the status of the hopper after payout is complete and to see if any errors occurred.

3.123 Header 132 - Emergency stop value

Transmitted data : <none>
Received data : [payout value remaining LSB] [payout value remaining MSB]

This command immediately halts the payout sequence and reports back the value of coins which failed to be paid out.

3.124 Header 131 - Request hopper coin value

Transmitted data : [coin type]
Received data : 6 x ASCII chars
[coin value LSB] [coin value MSB]

[coin type]
1 to N
(N = number of different coin types that can be dispensed by the hopper)

This command returns the name of the coin as well as the coin value in binary.

Refer to Appendix 3.1 for coin naming information.

3.125 Header 130 - Request indexed hopper dispense count

Transmitted data : [coin type]
Received data : [count 1] [count 2] [count 3]

[coin type]
1 to N
(N = number of different coin types that can be dispensed by the hopper)

The dispense counter records the number of each coin type dispensed by the hopper.

count 1 = LSB, count 3 = MSB

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

3.126 Header 129 - Read barcode data

Transmitted data : <none>
Received data : ASCII or ACK

When bill event code 20 is polled by a host system talking to a bill validator, a coupon has been inserted with a conforming barcode pattern. This barcode can be read as an ASCII string. The barcode data will remain (and can be read at any time) until a new coupon is inserted and another bill event code 20 is polled.

The ASCII data may contain any alpha-numeric characters applicable to the barcode formats supported by the device. For example, the interleaved 2 of 5 format returns 18 numeric characters e.g. ‘213866515856379505’. The interpretation of the barcode is entirely down to the host system – most barcodes are linked to back office systems.

If the barcode format supports a checksum character then this should be returned as part of the ASCII data.

If a valid barcode is not decoded on the last bill insertion then the response to this command is simply an ACK.

3.127 Header 128 - Request money in

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

4 bytes of data are returned, LSB first. These bytes combined represent a 32-bit unsigned integer in the range 0 to 4,294,967,295. This is a monetary value for the changer currency.

32-bit value in decimal = [value 1] + 256 * [value 2] + 65536 * [value 3] + 16777216 * [value 4]

This command reports the total value of coins entered into the changer. Units are the lowest value denomination for that currency e.g. cents or pence.

3.128 Header 127 - Request money out

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

4 bytes of data are returned, LSB first. These bytes combined represent a 32-bit unsigned integer in the range 0 to 4,294,967,295. This is a monetary value for the changer currency.

32-bit value in decimal = [value 1] + 256 * [value 2] + 65536 * [value 3] + 16777216 * [value 4]

This command reports the total value of coins paid out of the changer. Units are the lowest value denomination for that currency e.g. cents or pence.

3.129 Header 126 - Clear money counters

Transmitted data : <none>
Received data : ACK

Clears the ‘money in’ and ‘money out’ counters as reported by the ‘Request money in’ and ‘Request money out’ commands. This can be done at any time to reset the baseline for these counters.

3.130 Header 125 - Pay money out

Transmitted data : [value 1] [value 2] [value 3] [value 4]
Received data : ACK

4 bytes of data are sent, LSB first. These bytes combined represent a 32-bit unsigned integer in the range 0 to 4,294,967,295. This is a monetary value for the changer currency.

32-bit value in decimal = [value 1] + 256 * [value 2] + 65536 * [value 3] + 16777216 * [value 4]

The command pays a total value of coins out of the changer. The controller decides which combination of hoppers to use.

An ACK is returned by the changer if the command is received without error. In the event of an ACK not being received it is recommended that the event counter returned by the 'Verify money out' command is checked to ensure that a duplicate payment is not being made.

3.131 Header 124 - Verify money out

Transmitted data :	<none>
Received data :	[event counter]
	[paid 1] [paid 2] [paid 3] [paid 4]
	[unpaid 1] [unpaid 2] [unpaid 3] [unpaid 4]

event count = 1 to 255. This simple byte counter is used to identify a new money out completion event. On completion of a payout the paid and unpaid counters are updated and the event counter incremented. The value 0 is a special case and means a reset or power-down cycle has occurred and the paid and unpaid counters will have been cleared.

So the event counter values after a power-up are 0, 1, 2, 3… 253, 254, 255, 1, 2, 3… 253, 254, 255, 1, 2, 3…

Each paid and unpaid counter is an unsigned 32-bit integer returned LSB first.
32-bit paid = [paid 1] + 256 * [paid 2] + 65536 * [paid 3] + 16777216 * [paid 4]
32-bit unpaid = [unpaid 1] + 256 * [unpaid 2] + 65536 * [unpaid 3] + 16777216 * [unpaid 4]

This command confirms the value of coins paid out and the value of coins not paid out after the last pay command. Coins paid may be less than the requested value if a fault or error occurs, or the hoppers run out of change. The event counter will increment on completion of each new payout request.

3.132 Header 123 - Request activity register

Transmitted data :	<none>
Received data :	[activity register 1] [activity register 2]

[activity register 1]
B0: Singulator running
B1: Escalator / Conveyor running
B2: Processing money in
B3: Processing money out
B4: Fault detected
B5: Avalanche detected
B6: Changer initialising
B7: Entry flap open

[activity register 2]
B0: Continuous rejects
B1: Hopper configuration change
B2: Reject divert active
B3: Exit cup full
B4: Non-fatal fault detected
B5: {Reserved}
B6: {Reserved}
B7: {Reserved}

This command reports the current activity status of the Changer.

The Changer cannot accept system configuration and operating commands until the ‘Changer initialising’ flag is cleared.

All hopper money is considered paid when the ‘Processing money out’ flag clears. At this point the pay cycle can be verified.

3.133 Header 122 - Request error status

Transmitted data : <none>
Received data : [device no.] [fault code]

[device no.]
1 - Hopper 1
2 - Hopper 2
3 - Hopper 3
4 - Hopper 4
5 - Hopper 5
6 - Hopper 6
7 - Hopper 7
8 - Hopper 8
100 - Coin Acceptor
250 - Cashbox
255 - System

[fault code]
1 - hopper empty (requires refilling)
2 - hopper jam (remove hopper shelf and clear jam)
3 - hopper fraud attempt (alert security)
4 - hopper fault (service callout)
101 - coin acceptor jam (remove coin acceptor and clear jam)
102 - coin acceptor fraud attempt (alert security)
103 - coin acceptor fault (service callout)
104 – coin acceptor to manifold opto fault (check connector)
251 - cashbox full (empty cashbox)
252 - cashbox missing (insert cashbox)
255 - other

If there is no error then [0] [0] is returned.

If [255] [255] is returned then there is a general system fault which can be diagnosed further with header 232, 'Perform self-check'.

Errors are reported in PRIORITY order. A device with a fault or jam is reported before a lowest-value empty hopper.

3.134 Header 121 - Purge hopper

Transmitted data : [hopper no.] [count]
Received data : ACK

[hopper no.]
1 to max. no. of hoppers to pay change. Which coin is in which hopper can be determined by the 'Request hopper balance' command.
255 - purge entire system

[count]
0 - purge hopper until empty
1 to 255 - purge hopper by this number of coins

This command can be used to completely empty a specified hopper. The hopper motor is run until no coins are seen on the exit optos – at that point it is assumed the hopper is empty. The low level sensor is ignored.

It is possible to purge a smaller number of coins by using a non-zero count value.

If a hopper no. of 255 is sent then the entire system is purged of coins, regardless of count value. This will empty each hopper in turn and also purge all coins from the singulator and escalator belts.

3.135 Header 120 - Modify hopper balance

Transmitted data : [hopper no.] [count LSB] [count MSB]
Received data : ACK

This command can be used as part of the REFILL operation to initialise the hopper counters. Alternatively, hoppers can be refilled via the coin acceptor – in this case the counters will be updated automatically.

[hopper no.]
1 to max. no. of hoppers to pay change. Which coin is in which hopper can be determined by the 'Request hopper balance' command.

[count]
unsigned 16-bit integer

The count is the number of coins, not the coin value.

3.136 Header 119 - Request hopper balance

Transmitted data : [hopper no.]
Received data : [coin ID 1] [coin ID 2] [coin ID 3] [coin ID 4] [coin ID 5]
[coin ID 6] [count LSB] [count MSB]

[hopper no.]
1 to max. no. of hoppers to pay change

[coin ID]
The name of the coin in standard ccTalk format e.g. US025A.
`<2-letter country code> <3-digit value> <1-letter issue code>`

[count]
unsigned 16-bit integer

The count is the number of coins in the hopper, not the coin value.

This command returns the name and number of coins in the specified hopper. Note that this is an estimate of the number of coins based on counting coins in and out of the hopper. The changer does not know if coins are removed from the hopper bowl manually or added independently of the coin acceptor.

It is possible for the same coin to be in more than 1 hopper when hoppers are combined to increase capacity at the expense of coin types. The configuration of the changer allows for single coin sorting to multiple hoppers.

3.137 Header 118 - Modify cashbox value

Transmitted data : [value 1] [value 2] [value 3] [value 4]
Received data : ACK

4 bytes of data are sent, LSB first. These bytes combined represent a 32-bit unsigned integer in the range 0 to 4,294,967,295. This is a monetary value for the changer currency.

32-bit value in decimal = [value 1] + 256 * [value 2] + 65536 * [value 3] + 16777216 * [value 4]

This command can be used to clear (or preset) the value returned by the ‘Request cashbox value’ command. If the cashbox is emptied then this command can be used to update the changer.

3.138 Header 117 - Request cashbox value

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

4 bytes of data are returned, LSB first. These bytes combined represent a 32-bit unsigned integer in the range 0 to 4,294,967,295. This is a monetary value for the changer currency.

32-bit value in decimal = [value 1] + 256 * [value 2] + 65536 * [value 3] + 16777216 * [value 4]

This command returns the value of coins in the changer cashbox. Note that this is an estimate of the value based on counting coins in. The changer does not know if coins are removed from the cashbox manually or added independently of the coin acceptor.

3.139 Header 116 - Modify real time clock

Transmitted data : [value 1 – LSB] [value 2] [value 3] [value 4 – MSB]
Received data : ACK

This command sets a real-time clock using the UNIX **`time_t`** value which stores the number of seconds since 01/01/1970 00:00:00, as a 4 byte signed integer.

3.140 Header 115 - Request real time clock

Transmitted data : <none>
Received data : [value 1 – LSB] [value 2] [value 3] [value 4 – MSB]

This command reads a real-time clock as a UNIX **`time_t`** value which stores the number of seconds since 00:00:00 01/01/1970, as a 4 byte, signed integer.

3.141 Header 114 - Request USB id

Transmitted data : <none>
Received data : [VID – LSB] [VID – MSB] [PID – LSB] [PID – MSB]

Some implementations of ccTalk run over USB on a virtual COM port. The USB port enumerates according to the VID (Vendor id), the PID (Product id) and the USB serial number string. This command provides the host machine with the 16-bit word values for VID & PID at an application level rather than at a device driver level. Obviously before any ccTalk commands can be sent to the peripheral it must already have enumerated correctly with these codes and so they are provided for convenience only. The VID code is provided by the USB Implementers Forum for peripheral manufacturers (http://www.usb.org/home). The PID code should be unique to each USB product from a given manufacturer.

3.142 Header 113 - Switch baud rate

Transmitted data : [baud rate operation] [baud rate code]
Received data : ACK or NAK or [baud rate code]

[baud rate operation]
0 – request baud rate in use
1 – switch baud rate to new value
2 – request maximum baud rate supported
3 – request support for new baud rate

[baud rate code]
0 – 4800
1 – 9600 (the ccTalk default)
2 – 19,200
3 – 38,400
4 – 57,600
5 – 115,200
6 – 230,400
7 – 460,800
8 – 512,000
9 – 921,600
10 – 1,000,000
18 – 1,843,200
20 – 2,000,000
30 – 3,000,000

The standard or default ccTalk baud rate for multi-drop operation is 9600 baud. Some ccTalk peripherals are produced with custom baud rates (either faster or slower) and this should be documented clearly in the product manual. This command is used to allow dynamic switching of baud rates on a ccTalk bus where supported. If there is one ccTalk peripheral on the bus this is straightforward to do. Where there are multiple devices, the host must ensure that **all peripherals are operating at the same baud rate**. In cases where the peripheral mix is indeterminate then the host can also use this command to find a maximum common baud rate **before** switching to it.

3.142.1. Baud Rate Switching Examples

(a)
Transmitted data : [0] [0]
Received data : [1]

The baud rate in use is requested (the baud rate code field is ignored and is sent as zero). The return code is 1 = 9600 baud.

Note that this command is included for completeness and is somewhat redundant as unless the host sends the command at the correct baud rate there would be no reply anyway ! It may be useful if you do not know what baud rate code number represents that baud rate though. You could also use it as a handshake-verify after switching the baud rate rather than a ‘Simple poll’.

(b)
Transmitted data : [1] [9]
Received data : ACK

The baud rate has been switched to code 9 = 921,600. Note that the ACK must be returned at the current baud rate before switching to the new one to give the host time to reconfigure its UART registers. If the requested baud rate is not supported then the peripheral should return a NAK.

(c)
Transmitted data : [2] [0]
Received data : [10]

The maximum baud rate supported is requested (the baud rate code field is ignored and is sent as zero). The return code is 10 = 1,000,000 baud.

(d)
Transmitted data : [3] [10]
Received data : ACK

Support for a baud rate of code 10 = 1,000,000 is requested. An ACK is returned if this baud rate is supported, a NAK otherwise. **There is no actual change in baud rate**.

3.142.2. Baud Rate Switching Rules

When switching the baud rate with operation [1], the broadcast address can be used to switch all the peripheral baud rates simultaneously. All peripherals will reply with an ACK at the current baud rate which is likely to generate a host receive error as there will be start bit timing differences but this can be ignored and each peripheral checked later for a successful switch.

A problem arises if a switch to a higher baud rate produces intermittent or no communication at all. The peripheral could even report that it supports a baud rate of 1,000,000 but due to poor bus wiring, once switched it can no longer function. For this reason there needs to be a method of restoring the default baud rate of 9600.

3.142.3. Baud Rate Switching Recovery

The following methods are suggested but are not mandatory. Refer to the product manual for any recovery mechanisms that have been implemented.

3.142.3.1. Power Cycling

There should always be a power cycle recovery mechanism i.e. changes made to a peripheral's baud rate should never be made permanent. Removing power to the ccTalk bus should return all peripherals to the default baud rate. However, cycling power under host control is not always easy to do.

3.142.3.2. Software Reset

A software reset should restore the default baud rate in the same way as cycling the power. By using the broadcast address all peripherals can be reset simultaneously without disconnecting the power supply. However, if a peripheral is not communicating properly at the switched baud rate then this command is unlikely to work and another recovery mechanism is needed.

3.142.3.3. Autonomous Recovery

If the peripheral receives excessive communication errors on its own bus address (framing errors, checksum errors…) within a certain time window it could change its own baud rate back to 9600. Also, if no valid command is seen from the host within say 10s of a baud rate switch then the peripheral should switch to the default. If the host cannot communicate with a peripheral despite a number of retries it would presumably retry at 9600 as part of the recovery process (it could first send a broadcast reset to switch all working peripherals back to 9600).

3.142.3.4. Hardware Reset

If the ccTalk bus supports a hardware reset line (see ccTalk connectors type 5 & 8) then the host can pull this line low to reset all peripherals to the default baud rate.

3.142.3.5. Data Line Recovery

The host could pull the data line low for a certain length of time (a UART breaking condition) and those peripherals which could detect this condition could switch back to the default baud rate. If the host switches to 600 baud and sends out a 0x00 character then a pulse 15ms long is produced on the data line. A 10ms pulse could trigger a peripheral reset.

3.143 Header 112 - Read encrypted events

Transmitted data :	[challenge 1]
Received data :	[CRC checksum LSB] [random 1] [event counter]
	[result 1A] [result 1B]
	[result 2A] [result 2B]
	[challenge 1]
	[random 2]
	[result 3A] [result 3B]
	[result 4A] [result 4B]
	[result 5A] [result 5B]
	[CRC checksum MSB]

The command description here is a brief outline. For full implementation details with worked examples please contact Money Controls. The appropriate documentation will be sent out to you.

The 2 blocks of return data are DES-encrypted by the peripheral. The host machine decrypts them with the current DES key and confirms the validity of the challenge byte and CRC checksum.

This command can be used on both coin acceptors and bill validators when an encrypted response is required. It replaces header 229, ‘Read buffered credit or error codes’ and header 159, ‘Read buffered bill events’.

[challenge 1]
1 byte of random challenge data is generated by the host. This helps prevent replay attacks where the DES key is still secret but responses are stored and injected onto the bus later.

[random N]
2 bytes of random data generated by the peripheral to mix up the plaintext.

[CRC]
A 16-bit CRC checksum is calculated for the received data. This is the same checksum algorithm as the protocol layer and provides a quick and easy hashing algorithm to verify the decryption.

The checksum is calculated on 14 bytes from [random 1] to [result 5B].

[event counter]
The event counter works in the same way as header 229, ‘Read buffered credit or error codes’ or header 159, ‘Read buffered bill events’.

[result N]
The event buffer works in the same way as header 229, ‘Read buffered credit or error codes’ or header 159, ‘Read buffered bill events’. The result buffer contains a running history of the last 5 event codes; each code comprising 2 bytes.

3.144 Header 111 - Request encryption support

Transmitted data : [170] [85] [0] [0] [85] [170]
Received data : [protocol level] [command level]
[protocol key size]
[command key size] [command block size]
[trusted mode]
[BNV2 | BNV1] [BNV4 | BNV3] [BNV6 | BNV5]
[DES1] [DES2] [DES3] [DES4]
[DES5] [DES6] [DES7] [DES8]

The command description here is a brief outline. For full implementation details with worked examples please contact Money Controls. The appropriate documentation will be sent out to you.

This command always works unencrypted even if protocol level encryption is in place. Therefore a host machine can always identify the type of encryption being used by a peripheral.

Protocol level encryption is used at the packet level on every single message between the host machine and peripheral. Command level encryption is used to protect the data payload of selected commands only and these vary with the type of peripheral..

[protocol level]
0 – no encryption
1 – ccTalk Serial Protocol Encryption Standard 1.2

[command level]
0 – no encryption
11 – Serial Hopper Encryption Standard CMF1-1 (SCH2 L1 encryption)
12 – Serial Hopper Encryption Standard CMF1-2 (SCH3 L2 encryption)
13 – Serial Hopper Encryption Standard CMF1-3 (SCH3E L3 encryption)
21 – Serial Hopper Encryption Standard CMF2-1 (Combi encryption)
101 – DES Encryption
102 – AES Encryption (future support only)
103 – Triple DES Encryption (future support only)

Refer to the product manual for a list of which commands are encrypted and how exactly this is done.

[protocol key size]
24

The ccTalk encryption algorithm uses a 6 digit key stored in 3 bytes, equivalent to 24 bits (the actual bit range is a lot less than this).

[command key size]
0 – 256 bits or unused
64
128
192

Note that although DES will report a 64 bit key, only 56 bits are used in the encryption algorithm. Traditionally 8 bits are used for parity checking.

[command block size]
0 – unused
64
128

DES uses a block size of 64 bits.

[trusted mode]
0 – normal operating mode
255 – trusted key exchange mode

In *trusted key exchange mode*, the values of the keys follow; otherwise **zero bytes** are returned.

Entering this mode requires a physical link or equivalent on the product i.e. it cannot be entered remotely through the ccTalk bus for obvious security reasons.

[BNV]
3 bytes for the ccTalk encryption algorithm key. It was used originally on banknote validators, hence 'BNV' key.

These bytes can be cleared to zero if there is no protocol level encryption support.

[DES]
8 bytes for the DES key.

3.145 Header 110 – Switch encryption key

Transmitted data :	[old 1] [new 1] [old 2] [new 2] [old 3] [new 3] [old 4] [new 4] [old 5] [new 5] [old 6] [new 6] [old 7] [new 7] [old 8] [new 8]
Received data :	ACK

The command description here is a brief outline. For full implementation details with worked examples please contact Money Controls. The appropriate documentation will be sent out to you.

When command level encryption such as DES is being used, this command can rotate the DES key to a new value. The old key and the new key are encrypted with the current key (= old key) and sent to the peripheral for storing. If the old key does not match the current key then there is no response.

Key verification can be done by making the new key equal the old key. The peripheral does not store any data but returns an ACK if the key is correct. This allows the host machine to confirm a key when a new peripheral is installed prior to it actual use which may be many hours later.

The 2 blocks of transmitted data are DES-encrypted by the host machine. The peripheral decrypts the 2 blocks. For performance reasons and lifetime issues with EEPROM storage in the peripheral it is suggested that the host machine changes the DES key no frequently than once every 4 hours and preferably every 24 hours.

3.146 <u>Header 109 – Request encrypted hopper status</u>

Transmitted data :	[challenge 1] [challenge 2] [challenge 3]
Received data :	[CRC checksum LSB] [challenge 1] [event counter]
	[payout coins remaining] [last payout : coins paid]
	[last payout : coins unpaid] [random 1]
	[challenge 2]
	[challenge 3]
	[hopper status register 1] [hopper status register 2]
	[hopper status register 3] [level status]
	[random 2] [random 3]
	[CRC checksum MSB]

The command description here is a brief outline. For full implementation details with worked examples please contact Money Controls. The appropriate documentation will be sent out to you.

The 2 blocks of return data are DES-encrypted by the peripheral. The host machine decrypts them with the current DES key and confirms the validity of the 3 challenge bytes and CRC checksum.

[challenge N]
3 bytes of random challenge data are generated by the host. This helps prevent replay attacks where the DES key is still secret but responses are stored and injected onto the bus later.

[random N]
3 bytes of random data generated by the peripheral to mix up the plaintext.

[CRC]
A 16-bit CRC checksum is calculated for the received data. This is the same checksum algorithm as the protocol layer and provides a quick and easy hashing algorithm to verify the decryption.

The checksum is calculated on 14 bytes from [challenge 1] to [random 3].

[event counter] [payout coins remaining]
[last payout : coins paid] [last payout : coins unpaid]

This is the same data as returned by header 166, ‘Request hopper status’.

[hopper status register 1] [hopper status register 2] [hopper status register 3]

This is the same data as returned by header 163, ‘Test hopper’.

[level status]

This is the same data as returned by header 217, ‘Request payout high / low status’.

3.147 Header 108 – Request encrypted monetary id

Transmitted data :	[position] [challenge 1]
Received data :	[CRC checksum LSB] [position]
	[C1] [C2] [C3] [SF] [DP]
	[challenge 1]
	[random 1]
	[V1] [V2] [V3] [V4]
	[IL] [IN]
	[CRC checksum MSB]

The command description here is a brief outline. For full implementation details with worked examples please contact Money Controls. The appropriate documentation will be sent out to you.

The 2 blocks of return data are DES-encrypted by the peripheral. The host machine decrypts them with the current DES key and confirms the validity of the position byte, challenge byte and CRC checksum.

This command can be used on both coin acceptors and bill validators when an encrypted response is required. It replaces header 184, ‘Request coin id’, header 157, ‘Request bill id’ and header 156, ‘Request country scaling factor’.

[challenge 1]
1 byte of random challenge data is generated by the host. This helps prevent replay attacks where the DES key is still secret but responses are stored and injected onto the bus later.

[random 1]
1 byte of random data generated by the peripheral to mix up the plaintext.

[CRC]
A 16-bit CRC checksum is calculated for the received data. This is the same checksum algorithm as the protocol layer and provides a quick and easy hashing algorithm to verify the decryption.

The checksum is calculated on 14 bytes from [position] to [IN].

[position]
This is the position or index of the coin or bill within the currency specification. It always start from 1 and is typically in the range 1 to 16 or 1 to 64.

[C1] [C2] [C3]
This is the country or currency code in ASCII.

2 formats are supported…

ISO 3166-1-A2 country code and ISO 4217 currency code (not ISO 3166-1-A3).

ISO 3166 requires 2 letters, ISO 4217 requires 3 letters.

If ISO 3166 is being used then the first character [C1] is ‘#’.

ISO 3166 is still the default and recommended coding system for all ccTalk peripherals.

[SF]
Scaling Factor.
This is a number between 0 and 255.

This is the scaling factor for this monetary item in powers of 10, relative to the base or minor currency unit e.g. cents or pence.

The common values are 0, 1, 2, 3 and 4.
0 = value x 1
1 = value x 10
2 = value x 100
3 = value x 1000
4 = value x 10,000

[DP]
Decimal Places.
This is a number between 0 and 255.

The common values are 0 and 2.

A value of 100 with no decimal places would be displayed as 100 and with 2 decimal places as 1.00

The DP field is used primarily for the machine display of bill insertion values.

[V1] [V2] [V3] [V4]
Value in ASCII

e.g. 0000 to 9999

By ccTalk convention, coin acceptors report values in 3 digits and bill validators in 4 digits. However, using this encrypted command all values are reported in 4 digits with a leading zero digit if necessary.

[IL]
Issue Level in ASCII

This is a letter between ‘A’ and ‘Z’.

[IN]
Issue Number in ASCII

This is a number between ‘1’ and ‘9’.

The standard issue number progression is A1 to Z1 and then A2 to Z2 etc.

If there is no programmed bill at the specified position then a blank field designator should be returned.

```
CCC  = ... ( ASCII code 046 decimal )
VVVV = ....
IL   = .
IN   = .
SF   = 0
DP   = 0
```

3.148 Header 4 - Request comms revision

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

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

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

The ccTalk release number is used to track changes to the serial comms library in that particular product.

For example, the first release of a product conforming to the specification document at issue 4.6 should return [1] [4] [6]

3.149 Header 3 - Clear comms status variables

Transmitted data : <none>
Received data : ACK

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

3.150 Header 2 - Request comms status variables

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

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

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

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

This counter provides a mechanism for host software to determine the size of the slave receive buffer. A special ‘Simple poll’ command can be sent padded out with 252 bytes of dummy data. The size of the receive buffer = 257 - [rx bytes ignored] assuming that the ‘bytes ignored’ is non-zero and that the slave device stores the entire message packet.

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

A host machine could make use of this command to evaluate the amount of electrical noise on the serial bus. It could check these variables every hour for instance.

3.151 Header 1 - Reset device

Transmitted data : <none>
Received data : ACK

This command forces a *soft* reset in the slave device. It is up to the slave device what action is taken on receipt of this command and whether any internal house-keeping is done. The action may range from a jump to the reset vector to a forced physical reset of the processor and peripheral devices. This command is worth trying before a hard reset (or power-down where there is no reset pin) is performed.

The slave device should return an ACK immediately prior to resetting and allow enough time for the message to be sent back in full.

The host device should wait an appropriate amount of time after issuing a 'Reset device' command before sending the next command. Refer to the product manual for the reset initialisation time.