Part 3 - Contents

1 Appendix 1 - ccTalk Command Cross Reference

1 - Core commands
2 - Core Plus commands
3 - Multi-drop commands
C - Coin Acceptor commands
P - Payout commands (for serial hoppers)
B - Bill Validator commands
N - Changer / Escrow commands

Header	Function	1	2	3	C	P	B	N
255	Factory set-up and test							
254	Simple poll	1						
253	Address poll			3				
252	Address clash			3				
251	Address change			3				
250	Address random			3				
249	Request polling priority				C		B	
248	Request status				C			
247	Request variable set				C	P	B	N
246	Request manufacturer id	1						
245	Request equipment category id	1						
244	Request product code	1						
243	Request database version				C			
242	Request serial number		2					
241	Request software revision		2					
240	Test solenoids				C			N
239	Operate motors						B	N
238	Test output lines				C		B	
237	Read input lines				C		B	N
236	Read opto states				C	P	B	N
235	Read DH public key		2					
234	Send DH public key		2					
233	Latch output lines				C		B	
232	Perform self-check				C		B	N
231	Modify inhibit status				C		B	N
230	Request inhibit status				C		B	N
229	Read buffered credit or error codes				C			
228	Modify master inhibit status				C		B	
227	Request master inhibit status				C		B	
226	Request insertion counter				C		B	
225	Request accept counter				C		B	
224	Request encrypted product id		2					
223	Modify encrypted inhibit and override registers				C			
222	Modify sorter override status				C			
221	Request sorter override status				C			

220	ACMI encrypted data		2					
219	Enter new PIN number				C	P		
218	Enter PIN number				C	P		
217	Request payout high / low status					P		
216	Request data storage availability		2					
215	Read data block				C	P	B	N
214	Write data block				C	P	B	N
213	Request option flags				C		B	
212	Request coin position				C			
211	Power management control							
210	Modify sorter paths				C			N
209	Request sorter paths				C			N
208	Modify payout absolute count					P		
207	Request payout absolute count					P		
206								
205								
204	Meter control							
203	Display control							
202	Teach mode control				C		B	
201	Request teach status				C		B	
200	ACMI unencrypted product id		2					
199	Configuration to EEPROM				C			
198	Counters to EEPROM				C			
197	Calculate ROM checksum		2					
196	Request creation date		2					
195	Request last modification date		2					
194	Request reject counter				C		B	
193	Request fraud counter				C		B	
192	Request build code	1						
191	Keypad control							
190								
189	Modify default sorter path				C			
188	Request default sorter path				C			
187	Modify payout capacity					P		
186	Request payout capacity					P		
185	Modify coin id				C			N
184	Request coin id				C			N
183	Upload window data				C			
182	Download calibration info				C			
181	Modify security setting				C		B	
180	Request security setting				C		B	
179	Modify bank select				C		B	
178	Request bank select				C		B	
177	Handheld function				C			
176	Request alarm counter				C			
175	Modify payout float					P		N
174	Request payout float					P		N
173	Request thermistor reading				C	P		
172	Emergency stop					P		

171	Request hopper coin					P		
170	Request base year		2					
169	Request address mode		2					
168	Request hopper dispense count					P		
167	Dispense hopper coins					P		
166	Request hopper status					P		
165	Modify variable set					P		N
164	Enable hopper					P		
163	Test hopper					P		
162	Modify inhibit and override registers				C			
161	Pump RNG					P		
160	Request cipher key					P		
159	Read buffered bill events						B	
158	Modify bill id						B	
157	Request bill id						B	
156	Request country scaling factor						B	
155	Request bill position						B	
154	Route bill						B	
153	Modify bill operating mode						B	
152	Request bill operating mode						B	
151	Test lamps						B	N
150	Request individual accept counter						B	
149	Request individual error counter						B	
148	Read opto voltages						B	
147	Perform stacker cycle						B	
146	Operate bi-directional motors						B	N
145	Request currency revision						B	
144	Upload bill tables						B	
143	Begin bill table upgrade						B	
142	Finish bill table upgrade						B	
141	Request firmware upgrade capability						B	N
140	Upload firmware						B	N
139	Begin firmware upgrade						B	N
138	Finish firmware upgrade						B	N
137	Switch encryption code		2					
136	Store encryption code		2					
135	Set accept limit				C			
134	Dispense hopper value					P		
133	Request hopper polling value					P		
132	Emergency stop value					P		
131	Request hopper coin value					P		
130	Request indexed hopper dispense count					P		
129	Read barcode data						B	
128	Request money in							N
127	Request money out							N
126	Clear money counters							N

125	Pay money out							N
124	Verify money out							N
123	Request activity register							N
122	Request error status							N
121	Purge hopper							N
120	Modify hopper balance							N
119	Request hopper balance							N
118	Modify cashbox value							N
117	Request cashbox value							N
116	Modify real time clock							N
115	Request real time clock							N
114	Request USB id		2					
113	Switch baud rate		2					
112	Read encrypted events				C		B	
111	Request encryption support	1						
110	Switch encryption key		2					
109	Request encrypted hopper status					P		
108	Request encrypted monetary id				C		B	
107	Operate escrow							N
106	Request escrow status							N
105	Data stream		2					
104	Request service status							N
103	Expansion header 4							
102	Expansion header 3							
101	Expansion header 2							
100	Expansion header 1							
99 to 20	*Application specific*							
19 to 7	*Reserved*							
6	BUSY message		2					
5	NAK message		2					
4	Request comms revision		2					
3	Clear comms status variables				C	P	B	
2	Request comms status variables				C	P	B	
1	Reset device		2					
0	*Return message*							

1.1 Core Commands

These are the commands which should be supported by all ccTalk peripherals. They allow the device at the address specified to be precisely identified, even if the rest of the command set is unknown.

Here is an example for a Money Controls product…

Command	Response
Simple poll	ACK
Request equipment category id	“Coin Acceptor”
Request product code	“SR3”
Request build code	“TSTD”
Request manufacturer id	“Money Controls”
Request encryption support	No encryption

1.2 Core Plus Commands

These commands are useful but not essential for all ccTalk peripherals. For instance, it can be useful to have an electronic serial number in the product but not all devices support this feature.

1.3 Multi-drop Commands

These commands are not needed if the host is connected to only one ccTalk peripheral, or if all the addresses on the bus are known in advance. Otherwise these commands are needed to perform address resolution and dynamic address changing.

1.4 Coin Acceptor Commands

This is the recommended command subset for all ccTalk peripherals which return ‘Coin Acceptor’ as the category identification.

Note that not all commands will be implemented by every product. The serial communication manual for the product concerned should provide a definitive command list. See Appendix 13 for a minimum acceptable implementation.

1.5 Payout Commands

This is the recommended command subset for all ccTalk peripherals which return ‘Payout’ as the category identification. This is the case for serial compact hoppers etc.

Note that not all commands will be implemented by every product. The serial communication manual for the product concerned should provide a definitive command list. See Appendix 13 for a minimum acceptable implementation.

1.6 Bill Validator Commands

This is the recommended command subset for all ccTalk peripherals which return ‘Bill Validator’ as the category identification.

Note that not all commands will be implemented by every product. The serial communication manual for the product concerned should provide a definitive command list. See Appendix 13 for a minimum acceptable implementation.

1.7 Changer Commands

This is the recommended command subset for all ccTalk peripherals which return ‘Changer’ as the category identification.

Changers are money-in, money-out coin recyclers comprising a coin acceptor, a number of hoppers and an internal coin transport system.

Note that not all commands will be implemented by every product. The serial communication manual for the product concerned should provide a definitive command list. See Appendix 13 for a minimum acceptable implementation.

Please refer to Table 8, ccTalk Packet Lengths, for the expected size of TX and RX data.

Appendix 2 - OSI 7-layer Network Model

The OSI 7-layer network model is of limited use for a simple control protocol such as ccTalk. Whereas the task of writing software for full-blown networking protocols is made simpler by having a structured and modular approach, ccTalk is usually written from scratch on each new platform as the entire code is only a couple of kilobytes in size. Any artificial layering would be counter-productive.

For PC-based host software, writing a ccTalk DLL or OCX is an elegant way of separating the low level serial communication packet drivers from the high level application software. A generic ccTalk message sender can be made available through a public interface.

However, for completeness...

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

In broad terms...

RS232 deals with layers 1 & 2.

ccTalk deals with layers 3 & 4.

An encryption layer is used on some peripherals where security is paramount.

2 Appendix 3 - Coin Types and Coin Values

The 'Read buffered credit or error codes' command returns a 'coin type' or 'coin position' which is typically a number between 1 and 16. The host machine needs to translate this number into a monetary value for accumulating towards game credit or price settings. The method was chosen because it is the closest match to the existing parallel credit system and minimises changes to the host software when converting from parallel to serial operation.

The link between the coin type and the coin name is determined by the 'coin specification' programmed into the coin acceptor either at the factory or by portable equipment supplied to customers and service centres. The product label usually details which coins are programmed into which positions.

An example of a product label for a coin acceptor is…

C435A	GB

1	2	3	4	5	6	7	8
1.00	0.50	0.20	0.10	TK	2.00	0.05	

The label shows that coins are programmed into positions 1 to 7 of a 16 coin acceptor. The upper bank (coins 9 to 16) is not used in this example.

When polling for serial credits, the following codes are obtained…

Code	Coin Name	Monetary Value
1	GB 1.00	100
2	GB 0.50	50
3	GB 0.20	20
4	GB 0.10	10
5	TK Token	25
6	GB 2.00	200
7	GB 0.05	5

The host machine can be programmed with these assignments in a look-up table. Any coin acceptor ordered from the factory must be programmed with the correct coin specification.

2.1 Automatic Coin Configuration

It is possible using ccTalk to have the coin type table automatically downloaded from the coin acceptor into the host machine during power-up initialisation.

For this to work, there needs to be support for the ‘Request coin id’ command.

Each coin position, for example 1 to 16, is interrogated for an ASCII identifier. This consists of 6 characters representing the coin name.

The identifier is made up as follows…

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

CC = Standard 2 letter country code e.g. GB for the U.K. (Great Britain)
VVV = Coin value in terms of the base unit appropriate to that country
I = Mint Issue. Starts at A and progresses B, C, D, E…

Refer to Appendix 10 for a list of country codes.

The country code for the ‘Euro’, the Common European currency, is ‘EU’.

If the country code is ‘TK’ then a token occupies this position rather than a coin. In this case the VVV field represents a token number in ASCII rather than a value which could change from one jurisdiction to another.

It is possible to have more than one mint issue in circulation at any particular time - for instance during a transition period from ‘old’ coins to ‘new’ coins. Serial coin acceptors can be programmed with both types and the ‘old’ coins inhibited by the host machine when they officially go out of circulation.

2.1.1 Unprogrammed Coins and Erased Coins

Any coin positions that have been left unprogrammed or have been subsequently erased from the coin acceptor should return a blank coin name. By convention this is 6 x ASCII dots (decimal code 46) rather than 6 x ASCII spaces (decimal code 32).

e.g.

```
Coin 1 = EU200A
Coin 9 = ......
```

Coin position 1 has a programmed coin. Coin position 9 is unprogrammed.

2.1.2 Teach Coins

Some coin acceptors allow new coins to be 'taught' by inserting a sample of them rather than have them factory programmed. In this case the convention is to use a token identifier with value zero.

e.g.
`Coin 16 = TK000A`

Coin 16 is a coin or token that has been taught. There is no way to identify the value of the coin or token from the identifier alone.

2.1.3 Encrypted Bill & Coin Identifiers

Refer to header 108, 'Request encrypted monetary id'.

The following conventions have been adopted.

2.1.3.1 Unprogrammed Channel (Blank)

`Channel Id = .......00`

2.1.3.2 Tokens

`Channel Id = #TK0000A1 or TKN0000A1`

2.1.3.3 Teach Coins

`Channel Id = TCH0000..`

The coin value is defined more fully as follows…

3 x ASCII Characters	Value
5m0	0.005
10m or .01	0.01
20m or .02	0.02
25m	0.025
50m or .05	0.05
.10	0.10
.20	0.20
.25	0.25
.50	0.50
001	1
002	2
2.5	2.5
005	5
010	10
020	20
025	25
050	50
100	100
200	200
250	250
500	500
1K0	1,000
2K0	2,000
2K5	2,500
5K0	5,000
10K	10,000
20K	20,000
25K	25,000
50K	50,000
M10	100,000
M20	200,000
M25	250,000
M50	500,000
1M0	1,000,000
2M0	2,000,000
2M5	2,500,000
5M0	5,000,000
10M	10,000,000
20M	20,000,000
25M	25,000,000
50M	50,000,000
G10	100,000,000

As can be seen from the table, a scientific notation is used to compress large and small coin values into 3 characters. The shaded area represents the identifiers used for the vast majority of countries.

Some examples…

GB010B - 2nd mint issue of the U.K. 10p coin
US100B - 2nd mint issue of the U.S.A. 1$ coin
US005A - 1st mint issue of the U.S.A. 5c coin

2.2 CVF

The Coin Value Format or CVF is a quick method for returning coin value rather than coin position in the ‘Read buffered credit or error codes’ command. It is a useful option when the ‘Request coin id’ command is not supported but only works for coin values in the range 1 to 1000 in standard increments.

The CVF is basically a method for compressing coin values into a single byte.

A CVF byte consists of a multiplier bit (bit 7) and a 7-bit data value…
[multiplier bit | data value]

If the multiplier bit is set then the data value is multiplied by 10. This allows a convenient way of transmitting credit codes as monetary values with a ratio between largest and smallest coins in excess of 1000 to 1. A CVF byte of 255 is reserved for a token of unspecified value.

Here are the most common CVF values…

Coin Value	CVF code
1	1
2	2
5	5
10	10
20	20
25	25
50	50
100	100
200	148
250	153
500	178
1000	228
Token	255

The CVF bytes 0 and 128 have no monetary value. The maximum CVF byte is 254 (255 is a token) which corresponds to a coin value of 1260.

To determine whether a coin acceptor has been programmed with CVF codes, read option flag 0 with the ‘Request option flags’ command.

1.1 BACTA Token Selection

Token acceptance in a coin acceptor can be handled by ccTalk in a number of different ways. The first method shown here is the BACTA industry standard for the UK and is recommended for new designs. The second method is optional and could lead to compatibility issues with existing gaming machine software.

2.2.1 Token Selection

Each game has 1 active token. The coin acceptor can be programmed with a number of different tokens but only one of them can be selected at any one time. The selected token is used in place of coin position 5 which historically (in the days of parallel coin acceptors) has always been the token. When a token is inserted into the coin acceptor and is validated as true, credit code 5 is stored in the event buffer. The host machine knows that a token has been inserted and assigns the correct monetary value to it.

Any coins programmed into coin position 5 are disabled in this mode as a token substitution has been made. A coin acceptor may typically have 6, 12 or 16 programmed tokens and a manual method of selecting which one to use with a DIP switch or rotary switch.

To maximise the benefits of serial operation it makes sense to have a serial command to select the token to use. The convention has been to use ccTalk header 177, ‘Handheld function’. This is a general purpose command which allows manual switch-selectable configuration options (literally selected by ‘hand’) to be replaced with a serial equivalent. In this case we define ‘function 1’ to be token selection across all coin acceptors which support it.

Header byte = 177
Data byte 1 = 1 (mode = 0, function = 1)
Data byte 2 = Token selected

For example…

To select token 1
TX = 002 002 001 177 001 **001** 072
RX = 001 000 002 000 253 - ACK

To select token 5
TX = 002 002 001 177 001 **005** 068
RX = 001 000 002 000 253 - ACK

The commands for reading coin identifiers (ccTalk header 184) are unsupported for tokens as there is no standardised database as yet for token descriptors. So requesting the coin identifier for position 5 will produce undefined results.

In some coin acceptor configurations, the coins are programmed as 2 banks. For instance a coin acceptor with 12 programmed coins may be seen as having 2 banks of 6 coins or a coin acceptor with 16 programmed coins may be seen as having 2 banks of 8 coins. One bank may contain standard security coins and the other bank high security coins. Or one bank may contain old coinage still in circulation and the other bank new coinage. Or each bank may have a different currency. In these cases it is preferable to substitute the token into the upper bank as well so if the second bank is selected (through inhibits) then the token is selected as well. In a 12 coin acceptor then the corresponding upper bank token position is 11 and in a 16 coin acceptor it is 13.

2.2.2 Tokens as Coins

In this alternative method the tokens are treated identically to coins and programmed into the coin space as part of the currency specification. The country code is set to 'TK' and the value field becomes an arbitrary catalogue number (as yet not standardised). A credit code is obtained in the same way as coins when the coin acceptor is polled. Tokens can fill 1, 2 or all of the available coin positions.

Token selection is made by use of inhibits rather than the method described above.

The disadvantage of this method is that adding tokens to a coin specification reduces the number of coins that can be programmed in as well. Some dual currency specifications such as GB & Euro require nearly all 16 coin positions to be available for coins rather than tokens.

3 Appendix 4 - Polled Serial Credit Timing for Coin Acceptors

For a coin acceptor, a key consideration of the serial protocol is how long it takes to read out new credit information. There are 2 commands which will be considered here, the 'Read last credit or error code' command (now obsolete) and the 'Read buffered credit or error codes' command.

Read last credit or error code (4800 baud, SLOW option)

Host sends 5 bytes : [2] [0] [1] [235] [18]
Slave returns 6 bytes : [1] [1] [2] [0] [coin position] [checksum]

Each byte takes 2ms @ 4800 baud
Assume no gap between host bytes (fired out fast)
Assume a slave command response time of 3ms
Assume a gap between slave bytes of 1ms

Overall message time = 10 + 3 + 17 = **30ms**

For a coin acceptor that can accept 5 coins per second, we must poll it every 200ms. This means on a multi-drop network, we can support 200 / 30 = **6** identical coin acceptors.

Read buffered credit or error codes (9600 baud, 5 event buffer)

Host sends 5 bytes : [2] [0] [1] [229] [18]
Slave returns 16 bytes : [1] [11] [2] [0] [events]
[result 1A] [result 1B] [result 2A] [result 2B] [result 3A] [result 3B]
[result 4A] [result 4B] [result 5A] [result 5B] [checksum]

Each byte takes 1ms @ 9600 baud
Assume no gap between host bytes (fired out fast)
Assume a slave command response time of 2ms
Assume a gap between slave bytes of 1ms

Overall message time = 5 + 2 + 31 = **38ms**

For a coin acceptor that can accept 5 coins per second, we must poll it every 1000ms (because we have a 5 event buffer). This means on a multi-drop network, we can support 1000 / 38 = **26** identical coin acceptors.

For multi-drop networks, the use of the buffered serial credit command gives much better performance and allows more slave devices to be networked together.

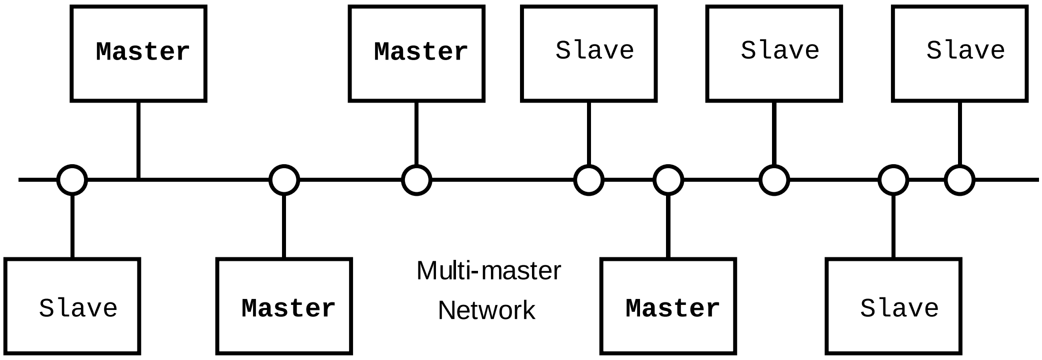
3.1 Polled Retries

Some protocols have a single-shot credit system such that each coin generates a single credit message that disappears after reading it. For this method to be secure, a retry mechanism needs to be in place at a low level to cope with an error in sending the credit information from the coin acceptor back to the host. If this data is corrupted and the credit information is re-read, the device may report no new credits !

The buffered credit system of ccTalk allows the message to be retransmitted repeatedly in the event of a communication error. The only limiting factor is the size of the event buffer as there will reach a point when new credits are over-written. A typical network configuration will allow plenty of retries before this could happen and if there is still a communication problem the coin acceptor could be shut down.

4 Appendix 5 - Multi-Master Applications

The ccTalk protocol is designed for single-master, multiple-slave applications**. It is not recommended that ccTalk is used in multi-master applications**. There are other control protocols more suited to multi-master operation. This section is included for interest only.



The addition of a source address field in ccTalk message packets allows any network device to talk to any other network device. If each ccTalk message packet on the bus is plucked out and examined, it knows where it is going and it knows where it has come from. So although the host machine could ask a coin acceptor for its serial number, a coin acceptor could in theory ask a bill validator for its escrow status.

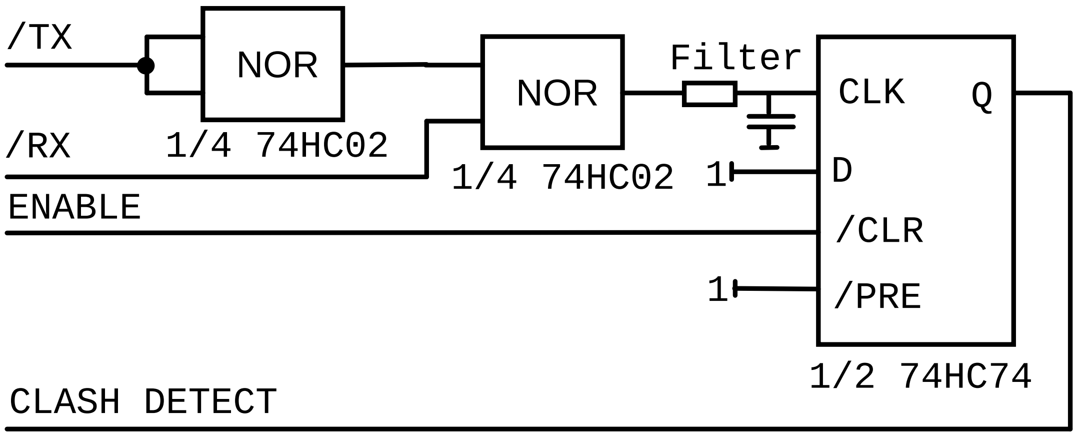
The biggest problem with this approach is network clashes. If 2 masters decide to transmit a message simultaneously or even near simultaneously then the message packets will collide. On a single data line this means any message bit could be 'scrambled'. Although this sounds bad, some networking protocols make use of this feature. If the data is scrambled then it is re-sent later, ideally after a random amount of time. With any luck the next retry will get through. This type of network is usually referred to as CSMA/CD, i.e. Carrier Sense Multiple Access with Collision Delay. There is a *chance* of a message collision which can be estimated for any degree of network loading. Clearly the more masters that exist on a network, the more frequently they transfer information and the longer the message packets, the less chance there is of a message getting through. When the network loading is low, the chances of collisions are so small that they can effectively be ignored and we have a *true* network. As loading increases, the number of retries goes up and eventually the network becomes unusable.

For ccTalk to detect a network clash, one or more of the following events must occur.
a) RS232 framing error (the stop bit was low not high)
b) 8-bit checksum error

Reliance on these conditions for money transactions would be represent a big security lapse as a collision would not always be detected. Messages could collide and transform themselves into different but seemingly authentic ones.

The addition of some simple electronics allows far more reliable data transfer in such situations. We can detect the condition whereby a device is transmitting a logic 1 but another device on the bus is transmitting a logic 0. Since a logic 0 on the serial bus is an overriding condition which cannot be changed by another ccTalk peripheral, this is the one illegal state we can detect. The circuit below clocks a latch (D-type flip-flop) when a logic 1 output by a device on the bus is forced low by another peripheral. The transmitting device would only enable the latch when sending data and would read back the clash detect signal immediately afterwards. If it is high then a collision has occurred (presumably due to another master on the system) and the device can re-send the message packet after a fixed or random delay. This system can be implemented by the host for sending a command to a slave device and by the slave device when replying with data.

A small filter can be included on the clock line to the latch to remove glitches due to transmission / reception delays.



The ability of a device to successfully talk to a slave on a multi-master network can be estimated with probability theory. Unlike a deterministic network with *time slots* that can be allocated according to priority, ccTalk is non-deterministic in that there is no way of knowing in advance whether a particular command will succeed.

Network Loading Equations

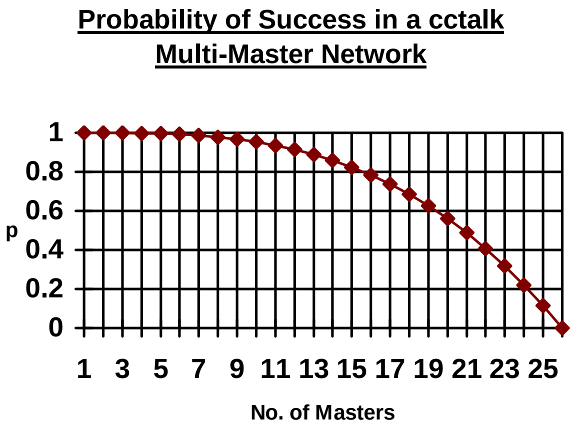
Define...
n = no. of masters on bus
f = frequency of communication (/s)
t = average length of message (s)
r = no. of retries before command aborted

When a device transmits a command, the chance of a collision = (n - 1) . f . t

This is a simple estimate based on the available time in each second during which a short command could be sent. Note that when there is only 1 master there is no chance of a collision.

Allowing for retries, the success of a particular command = 1 - ((n - 1) . f . t) ^ r

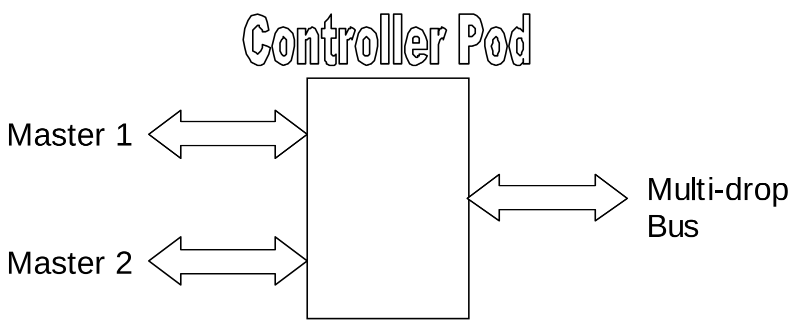
Here is an example for illustration :
Let... f = 1 (once per second), t = 40ms (typical command), r = 3 (3 goes max.)



As can be seen from the graph, the performance is fairly flat to start off with and then drops dramatically as the network limit approaches. In this example, the network can support 12 masters quite comfortably (i.e. > 90% success rate).

4.1 Arbitration Controllers

For a network which only requires 2 masters, another solution to the problem would be a small controller pod which isolates the 2 masters from each other. The controller



pod would arbitrate between the 2 masters and decide which one has access to the multi-drop bus. If both masters require access together, one of them could be delayed by replying with the ccTalk BUSY message.

5 Appendix 6 - Naming Convention

The following naming convention may be adopted in technical literature to indicate the exact ccTalk interface specification in use on the product. The idea is to allow some kind of serial communication to be set up given only this name and no product manual (the usual situation in engineering !).

The ccTalk feature list currently runs to 11 items…

ccTalk

b	**b**aud rate ÷ 100
p	interface **p**ort
v	supply **v**oltage
a	d**a**ta voltage
d	supply **d**irection
c	**c**onnector type
m	**m**aster / slave configuration
x	checksum type
e	**e**ncryption type
i	specification release - m**i**nor (previously the *level*)
r	specification **r**elease - major

b. The baud rate may be…

```
 48 –     4,800 baud
 96 –     9,600 baud ( default )
192 –    19,200 baud
384 –    38,400 baud
576 –    57,600 baud
115 –   115,200 baud
230 –   230,400 baud
460 –   460,800 baud
512 -   512,000 baud
921 –   921,600 baud
100 – 1,000,000 baud
184 - 1,843,200 baud
200 – 2,000,000 baud
300 – 3,000,000 baud
```

p. The interface port is defined as follows…

0 – open-collector interface (default)
1 – RS485 interface
2 – USB via SiLabs hardware & VCOM drivers
3 – USB via FTDI hardware & VCOM drivers

v. The supply voltage refers to the nominal power supply voltage to the product, specified in volts.

5 – 5V
12 – 12V (default)
24 – 24V
48 – 48V
It is assumed that all voltages are positive, regulated D.C.

a. The data voltage is the bus pull-up voltage when using an open-collector interface (not applicable on USB). Note that ccTalk always uses 0V as the active state (start bit condition) but the idle state voltage can be altered to suit the application.

3 – 3V3
5 – 5V (default)
12 – 12V
24 – 24V

It is assumed that for voltages other than 5V, the data voltage will usually track the supply voltage.

d. The supply direction is defined as follows…
0 – supply sink (default, an external power supply must be connected)
1 – supply source (can be used to power other peripherals)
2 – supply sink or source

c. The connector type is defined elsewhere in this document.
Additionally…
USB – standard USB type B connector

m. The master / slave configuration is defined as follows…
0 – slave device (default, only replies to ccTalk messages)
1 – master device (initiates ccTalk messages)
2 – master or slave device (manual switching)
3 – master or slave device (automatic switching)

x. The checksum type is defined as follows…
8 – addition checksum, byte (default)
12 – addition checksum, 16-bit word
16 – CRC CCITT checksum, 16-bit word

e. The encryption type is defined as follows…
0 – none (default)
1 – encryption type 1

i. For specification release - minor, use the **minor issue number** of this document.

On older ccTalk products, this number refers to the implementation level.

r. For specification release - major, use the **major issue number** of this document.

If a feature isn't specified then assume the default setting.

5.1 Money Controls Product Examples

SCH3
ccTalk b96.p0.v24.a5.d0.c8.m0.x8.e0.i4.r4

Expands as 9600 baud, open-collector, +24V supply, +5V data, supply sink, connector type 8, slave device, 8-bit checksum, no encryption, minor release 4, major release 4

SR5i
ccTalk b96.p0.v12.a5.d0.c5.m0.x8.e0.i2.r4

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 5, slave device, 8-bit checksum, no encryption, minor release 2, major release 4

SR3i
ccTalk b96.p0.v12.a5.d0.c7.m0.x8.e0.i2.r4

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 7, slave device, 8-bit checksum, no encryption, minor release 2, major release 4

C120S
ccTalk b48.p0.v5.a5.d0.c3.m0.x8.e0.i2.r4

Expands as 4800 baud, open-collector, +5V supply, +5V data, supply sink, connector type 3, slave device, 8-bit checksum, no encryption, minor release 2, major release 4

Lumina
ccTalk b96.p0.v12.a5.d0.c5.m0.x16.e1.i2.r4

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 5, slave device, CRC CCITT checksum, encryption type 1, minor release 2, major release 4

SCH2
ccTalk b96.p0.v24.a5.d0.c8.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +24V supply, +5V data, supply sink, connector type 8, slave device, 8-bit checksum, level 1, release 3

SR5
ccTalk b96.p0.v12.a5.d0.c5.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 5, slave device, 8-bit checksum, level 1, release 3

SR3
ccTalk b96.p0.v12.a5.d0.c7.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 7, slave device, 8-bit checksum, level 1, release 3

Condor Plus
ccTalk b96.p0.v12.a5.d0.c7.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +12V supply, +5V data, supply sink, connector type 7, slave device, 8-bit checksum, level 1, release 3

Serial Compact Hopper Mk1
ccTalk b96.p0.v24.a5.d0.c6.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +24V supply, +5V data, supply sink, connector type 6, slave device, 8-bit checksum, level 1, release 3

C435S
ccTalk b96.p0.v12.a12.d0.c5.m0.x8.i1.r3

Expands as 9600 baud, open-collector, +12V supply, +12V data, supply sink, connector type 5, slave device, 8-bit checksum, level 1, release 3

C435SR
ccTalk b48.p0.v12.a5.d0.c1.m0.x8.i1.r2

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

C120P
ccTalk b48.p0.v5.a5.d0.c3.m0.x8.i4.r2

Expands as 4800 baud, open-collector, +5V supply, +5V data, supply sink, connector type 3, slave device, 8-bit checksum, level 4, release 2

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

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

6 Appendix 7 - Flash Memory Support

Many processors now support flash uploading of code which has obvious advantages for manufacturing and field upgrades. If a product has a ccTalk serial interface then it makes sense to use the same connector for on-circuit flash re-programming.

Commands have been added into the BNV command set for flash programming - see headers 138 to 141.

Large flash memories can be hampered by a slow baud rate. 9600 baud is fine for control messages but slow for firmware upgrades. A 64K block of memory would take 1 minute 8 seconds to transfer without the associated protocol overheads. The net result could typically be 2 to 3 minutes for each 64K block.

Newer implementations of the protocol are now making use of the high-speed data transfer available over USB hardware. For example using 'ccTalk over USB' and VCOM drivers available from several chip manufacturers, firmware programming times can be cut down from minutes to seconds.

7 Appendix 8 - Address Clash Probability

A section for aficionados of probability theory !

Question : What is the probability of an address clash when ‘n’ devices are connected to the ccTalk bus with random addresses ?

Solution : This question crops up in many forms in probability theory, a particular favourite being how many people in a room before there is a better than evens chance of two people sharing the same birthday.

Like most problems, the solution is often easier when turned on its head. Instead of calculating how many devices could share an address, calculate how many outcomes there are of all the addresses being different.

Let total no. of addresses = m

Let no. of devices on bus = n

Assuming all addresses are different, the number of possible **permutations** we can have is…

m! / (m - n)!

! = Factorial, i.e. m! = m * (m-1) * (m-2) * (m-3) *… * 1

The total possible number of address permutations = m ^ n
This is ‘m’ possibilities for the first device multiplied by ‘m’ possibilities for the second device multiplied by… etc.

We now flip it back to find the chance of a clash (2 or more devices with the same address…)

p_{clash} = (m ^ n) - (m! / (m - n)!) / (m ^ n)

Rewriting…

p_{clash} = 1 - (m! / ((m - n)! * (m ^ n)))

For ccTalk, m = 254. There are 254 possible addresses as the broadcast address and host address are excluded.

7.1 Clash Results

n	p_{clash}
1	0.0000%
2	0.3937%
3	1.1780%
4	2.3452%
5	3.8831%
6	5.7751%
7	8.0009%
8	10.5363%
9	13.3541%
10	16.4242%
11	19.7146%
12	23.1915%
13	26.8203%
14	30.5657%
15	34.3928%
16	38.2672%
17	42.1559%
18	46.0274%
19	49.8522%
20	53.6034%
25	70.5071%
30	83.1874%
40	96.0981%

So for 3 serial compact hoppers attached to the bus, randomising their addresses would be successful a fraction less than 99 times out of a 100 first pass (the process could obviously be repeated). However, as the number of devices attached to the bus increase, there is a dramatic fall off in success rate. The break even point is 20 devices.

2. Appendix 9 - CRC Checksum Algorithm

7.2 Outline

The particular CRC checksum used in ccTalk is taken from the CRC-CCITT standard.

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

7.3 Example Command

To calculate the CRC protected message packets for the 'Reset device' command…

```
TX : [ 40 ] [ 0 ] [ crc_lsb ] [ 1 ] [ crc_msb ]
RX : [  1 ] [ 0 ] [ crc_lsb ] [ 0 ] [ crc_msb ]
```

The TX packet is to address 40 (bill validator) with header 1 (Reset device) and no data. The receive packet is to address 1 (host controller) with a null header and no data (the ACK message).

```
TX crc = CRC( 40, 0, 1 ) = 3F46 hex
RX crc = CRC(  1, 0, 0 ) = 3730 hex

TX crc_lsb = 46 hex, 70 dec
TX crc_msb = 3F hex, 63 dec
RX crc_lsb = 30 hex, 48 dec
RX crc_msb = 37 hex, 55 dec
```

Therefore, the completed message packets are…

```
TX : [ 40 ] [ 0 ] [ 70 ] [ 1 ] [ 63 ]
RX : [  1 ] [ 0 ] [ 48 ] [ 0 ] [ 55 ]
```

7.4 Algorithm in C++

7.4.1 Look-Up Table Method

```
void generate_crc_lookup_CCITT_A( void )

{
int  i;
BYTE z = 0;

for ( i = 0; i < 256; ++i )
        crc_lookup_table_CCITT_A[ i ] = calculate_crc_loop_CCITT_A( 1, &z, i << 8 );

}

WORD calculate_crc_lookup_CCITT_A( int l, BYTE *p, WORD seed )

{
int  i;
WORD crc = seed;

for ( i = 0; i < l; ++i )
        crc = ( crc << 8 ) ^ crc_lookup_table_CCITT_A[ ( crc >> 8 ) ^ p[ i ] ];

return crc;

}
```

7.4.2 Slower Method

```
WORD calculate_crc_loop_CCITT_A( int l, BYTE *p, WORD seed )

{
int  i, j;
WORD crc = seed;

for ( i = 0; i < l; ++i )
        {
        crc ^= ( p[ i ] << 8 );

        for ( j = 0; j < 8; ++j )
                {
                if ( crc & 0x8000 )
                        crc = ( crc << 1 ) ^ 0x1021; // 0001.0000 0010.0001 = x^12 + x^5
+ 1 ( + x^16 )
                else
                        crc <<= 1;
                }
        }

return crc;

}
```

7.5 Pre-Calculated Look-up Table

512 bytes

```
const unsigned short crc_ccitt_lookup[] =
{
0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

7.6 Verification Data

Random test data…

```
Data = 49 D5 F2 / CRC-CCITT Checksum = A6B3
Data = 2F BD 9D / CRC-CCITT Checksum = 90B2
Data = D9 53 D1 / CRC-CCITT Checksum = 7BB5

Data = 70 B8 D9 64 04 15 / CRC-CCITT Checksum = FB00
Data = 72 61 B9 4E D0 78 / CRC-CCITT Checksum = 93E3
Data = 63 FA D1 9F E6 19 / CRC-CCITT Checksum = 5BB3
```

3. Appendix 10 - Common Country Codes

Each country of the world has a 2 letter designator which conforms to ISO 3166-1-A2. Listed below are all the countries, split into ‘Europe’, ‘Rest of the World’ and ‘Islands’. Note that some serial protocols use ISO 4217 which identifies currencies rather than countries with a 3 letter code e.g. USD for U.S.A. Dollars.

3.1 Europe

Country	**3166-1-A2**
Albania	AL
€ Andorra	AD
€ Austria	AT
€ Belgium	BE
Bosnia Herzegovina	BA
Bulgaria	BG
Croatia	HR
Czech Republic	CZ
Denmark	DK
Estonia	EE
EURO	EU
€ Finland	FI
€ France	FR
Gibraltar	GI
€ Germany	DE
€ Greece	GR
Hungary	HU
Iceland	IS
€ Irish Republic	IE
Israel	IL
€ Italy	IT
Latvia	LV
Liechtenstein	LI
Lithuania	LT
€ Luxembourg	LU
Macedonia	MK
Moldova	MD
€ Monaco	MC
€ Montenegro	ME
€ Netherlands	NL
Norway	NO
Poland	PL
€ Portugal	PT
Romania	RO
€ San Marino	SM
Serbia	RS
~~Serbia & Montenegro~~	~~CS~~
Slovakia	SK
Slovenia	SI
€ Spain	ES
Sweden	SE
Switzerland	CH
Turkey	TR

United Kingdom	GB
€ Vatican City	VA

€ = Euroland country

3.2 Rest of the World

Country	3166-1-A2
Afghanistan	AF
Algeria	DZ
Angola	AO
Antarctica	AQ
Argentina	AR
Armenia	AM
Australia	AU
Azerbaijan	AZ
Bahrain	BH
Bangladesh	BD
Bhutan	BT
Belarus	BY
Belize	BZ
Benin	BJ
Bolivia	BO
Botswana	BW
Brazil	BR
Brunei	BN
Burkina Faso	BF
Burundi	BI
Cambodia	KH
Cameroon	CM
Canada	CA
Central African Rep.	CF
Chad	TD
Chile	CL
China	CN
Columbia	CO
Congo	CG
Costa Rica	CR
Cote D’Ivoire	CI
Djibouti	DJ
East Timor	TP
Ecuador	EC
Egypt	EG
El Salvador	SV
Eritrea	ER
Ethiopia	ET
Equatorial Guinea	GQ
French Guiana	GF
French Polynesia	PF
Gabon	GA
Gambia	GM
Georgia	GE
Ghana	GH
Greenland	GL

Guatemala	GT
Guinea	GN
Guinea-Bissau	GW
Guyana	GY
Honduras	HN
Hong Kong	HK
India	IN
Indonesia	ID
Iran	IR
Iraq	IQ
Japan	JP
Jordan	JO
Kazakhstan	KZ
Kenya	KE
Korea North	KP
Korea South	KR
Kuwait	KW
Kyrgyzstan	KG
Laos	LA
Lebanon	LB
Lesotho	LS
Liberia	LR
Libya	LY
Macau	MO
Malaysia	MY
Malawi	MW
Mali	ML
Mauritania	MR
Mexico	MX
Mongolia	MN
Morocco	MA
Mozambique	MZ
Myanmar	MM
Namibia	NA
Nepal	NP
New Zealand	NZ
Nicaragua	NI
Niger	NE
Nigeria	NG
Oman	OM
Pakistan	PK
Panama	PA
Papua New Guinea	PG
Paraguay	PY
Peru	PE
Philippines	PH
Puerto Rico	PR
Qatar	QA
Russia	RU
Rwanda	RW
Samoa	WS
Saudi Arabia	SA
Senegal	SN
Sierra Leone	SL

Singapore	SG
Somalia	SO
South Africa	ZA
Sudan	SD
Suriname	SR
Swaziland	SZ
Syria	SY
Tajikistan	TJ
Tanzania	TZ
Thailand	TH
Taiwan	TW
Togo	TG
Tunisia	TN
Turkmenistan	TM
Uganda	UG
Ukraine	UA
United Arab Emirates	AE
United States	US
Uruguay	UY
Uzbekistan	UZ
Venezuela	VE
West African States	XO
Western Sahara	EH
Vietnam	VN
Yemen	YE
Zaire	ZR
Zambia	ZM
Zimbabwe	ZW

3.3 Islands

Country	**3166-1-A2**
American Samoa	AS
Anguilla	AI
Antigua & Barbuda	AG
Aruba	AW
Bahamas	BS
Barbados	BB
Bermuda	BM
Bonaire	QQ
Bouvet Island	BV
Cape Verde	CV
Cayman Islands	KY
Christmas Island	CX
Cocos (Keeling) Islands	CC
Comoros	KM
Cook Islands	CK
Cuba	CU
Cyprus	CY
Dominica	DM
Dominican Republic	DO
East Caribbean	EA
Falkland Islands / Malvinas	FK

Faroe Islands	FO
Fiji	FJ
Jamaica	JM
Jersey	GB (JE)
Grenada	GD
Guadeloupe	GP
Guam	GU
Guernsey	GB (GG)
Haiti	HT
Heard & McDonald Islands	HM
Isle of Man	IM
Jamaica	JM
Kiribati	KI
Madagascar	MG
Maldives	MV
Malta	MT
Marshall Islands	MH
Martinique	MQ
Mauritius	MU
Mayotte	YT
Micronesia, Fed. States of	FM
Montserrat	MS
Nauru	NR
Netherlands Antilles	AN
New Caledonia	NC
Niue	NU
Norfolk Island	NF
Northern Mariana Islands	MP
Palau	PW
Pitcairn	PN
Reunion	RE
Sao Tome and Principe	ST
Seychelles	SC
Solomon Islands	SB
Sri Lanka	LK
St. Kitts & Nevis	KN
St. Helena	SH
St. Lucia	LC
St. Pierre & Miquelon	PM
St. Vincent & Grenadines	VC
Svalbard & Jan Mayen Islands	SJ
Tokelau	TK
Tonga	TO
Trinidad & Tobago	TT
Turks & Caicos	TC
Tuvalu	TV
Vanuatu	VU
Virgin Islands (GB)	VG
Virgin Islands (US)	VI
Wallis & Futuna	WF

ISO Web Reference:
http://www.iso.org/iso/iso-3166-1_decoding_table

7.7 Exceptions

Example token : TK830A

At Crane Payment Solutions - Money Controls we have adopted TK to designate a token. A token code number then follows the letters. Token code numbers have not yet been standardised across the industry. Note that in the ISO standard, TK is used for 'Tokelau'.

Some other Money Controls *historical anomalies*…

1. Aruba can use AA as well as AW
2. Ecuador can use ED as well as EC
3. Serbia & Montenegro used to be SX then CS. These countries now have separate identities.

For banknotes we have created a virtual country of 'BC' to represent barcode tickets and coupons.

```
BC0001A : Barcode 67mm x 14mm 2:1 ratio 2 of 5 interleaved
BC0001B : Barcode 84mm x 17mm 3:1 ratio 2 of 5 interleaved
BC0001C : Barcode 67mm x 27mm 2:1 ratio 2 of 5 interleaved
BC0002A : Jackpot Ticket
```

Although each casino prints its own tickets they have a number of common features which can be identified generically.

The use of XO for West African States is not officially recognised by ISO 3166. There is an ISO 4217 designation of XOF for the West African CFA franc, where CFA is the Communauté Financière d'Afrique (Financial Community of Africa). This covers eight independent states of West Africa: Benin, Burkina Faso, Côte d'Ivoire, Guinea-Bissau, Mali, Niger, Sénégal and Togo. The franc is nominally sub-divided into centimes but no centimes have ever been issued.

4. Appendix 11 - Coin Acceptor Messaging Example

4.1 Initialisation

This is a typical initialisation or enrolment process for a gaming machine. The idea is to confirm the ccTalk link to the coin acceptor is operational and that the device fitted is approved for use in this environment.

Simple poll = ACK (confirms a ccTalk device is operating at address 2)
```
TX = 002 000 001 254 255
RX = 001 000 002 000 253
```

Request equipment category id = ‘Coin Acceptor’
```
TX = 002 000 001 245 008
RX = 001 013 002 000 067 111 105 110 032 065 099 099 101 112 116 111 114 022
```

Request product code = ‘SR5i’
```
TX = 002 000 001 244 009
RX = 001 004 002 000 083 082 053 105 182
```

Request build code = ‘STD01   ’
```
TX = 002 000 001 192 061
RX = 001 008 002 000 083 084 068 048 049 032 032 032 073
```

Request manufacturer id = ‘Money Controls’
```
TX = 002 000 001 246 007
RX = 001 014 002 000 077 111 110 101 121 032 067 111 110 116 114 111 108 115
115
```

Request serial number = 12345678
```
TX = 002 000 001 242 011
RX = 001 003 002 000 078 097 188 143
```

Request software revision = ‘CRS-F1-V1.09’
```
TX = 002 000 001 241 012
RX = 001 012 002 000 067 082 083 045 070 049 045 086 049 046 048 057 026
```

Request comms revision = 1.4.2
```
TX = 002 000 001 004 249
RX = 001 003 002 000 001 004 002 243
```

4.2 Pre-Acceptance

Request coin id for coin 1 = 'GB200A'

```
TX = 002 001 001 184 001 067
RX = 001 006 002 000 071 066 050 048 048 065 155
```

Request coin id for coin 2 = 'GB100A'

```
TX = 002 001 001 184 002 066
RX = 001 006 002 000 071 066 049 048 048 065 156
```

Request coin id for coin 3 = 'GB050B'

```
TX = 002 001 001 184 003 065
RX = 001 006 002 000 071 066 048 053 048 066 151
```

Request coin id for coin 4 = 'GB020A'

```
TX = 002 001 001 184 004 064
RX = 001 006 002 000 071 066 048 050 048 065 155
```

Request coin id for coin 5 = 'Token '

```
TX = 002 001 001 184 005 063
RX = 001 006 002 000 084 111 107 101 110 032 214
```

Request coin id for coin 6 = 'GB010B'

```
TX = 002 001 001 184 006 062
RX = 001 006 002 000 071 066 048 049 048 066 155
```

Request coin id for coin 7 = '……'

```
TX = 002 001 001 184 007 061
RX = 001 006 002 000 046 046 046 046 046 046 227
```

Request coin id for coin 8 = '……'

```
TX = 002 001 001 184 008 060
RX = 001 006 002 000 046 046 046 046 046 046 227
```

We could also read the coin identifiers for coins 9 to 16 if we wanted to.

Modify inhibit status = ACK

```
TX = 002 002 001 231 255 255 022
RX = 001 000 002 000 253
```

All coins are now enabled for acceptance.

4.3 Credit Polling

Poll every 200ms and don't stop…

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 000 000 000 000 000 000 000 000 000 000 000 242
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 000 000 000 000 000 000 000 000 000 000 000 242
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 000 000 000 000 000 000 000 000 000 000 242
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 000 000 000 000 000 000 000 000 000 000 000 242
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 001 001 005 000 000 000 000 000 000 000 000 235

Event code has changed from 0 to 1 : Credit, coin 1 to sorter route 5
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 002 001 005 001 005 000 000 000 000 000 000 228

Event code has changed from 1 to 2 : Credit, coin 1 to sorter route 5
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 003 005 005 001 005 001 005 000 000 000 000 217

Event code has changed from 2 to 3 : Credit, coin 5 ( = token ) to sorter
route 5
```

Read buffered credit or error codes

```
TX = 002 000 001 229 024
RX = 001 011 002 000 004 000 001 005 005 001 005 001 005 000 000 215

Event code has changed from 3 to 4 : Error, Reject coin
```

Remember that the event code wraps from 255 to 1 as 0 is a special start-up value indicating a power-up or reset has occurred.

5. Appendix 12 - Italian Flavour Specification Changes

For Italian homologation 2003, the following proposals were put in place to guarantee product security. No ccTalk commands were changed but a small subset of commands were ‘switched off’ to prevent changes to configuration data. It is recommended that peripheral devices implementing the Italian flavour of ccTalk do not return any reply to these commands - not even a dummy ACK or a NAK.

SR3 is representative here of a 3.5 inch coin acceptor. SR5 is representative of a 5 inch coin acceptor.

Header	Function	SR3 Implementation	SR5 Implementation
216	Request data storage availability	No reply	OK
215	Read data block	No reply	OK
214	Write data block	No reply	OK
202	Teach mode control	No reply	No reply
201	Request teach status	No reply	No reply
185	Modify coin id	No reply	No reply
183	Upload window data	No reply	No reply
182	Download calibration info	No reply	No reply
181	Modify security setting	No reply	No reply
180	Request security setting	No reply	No reply
179	Modify bank select	No reply	No reply
178	Request bank select	No reply	No reply
177	Handheld function	No reply	No reply

The following commands were seen as ‘core’ to the Italian amusement market and **have to be implemented** by all peripherals. All other commands are optional.

Header	Function	ccTalk Class Designation
254	Simple poll	Core
253	Address poll	Multi-drop
252	Address clash	Multi-drop
251	Address change	Multi-drop
250	Address random	Multi-drop
249	Request polling priority	
246	Request manufacturer id	Core
245	Request equipment category id	Core
244	Request product code	Core
242	Request serial number	Core Plus
231	Modify inhibit status	
230	Request inhibit status	
229	Read buffered credit or error codes	
210	Modify sorter paths	
209	Request sorter paths	
192	Request build code	Core
184	Request coin id	
4	Request comms revision	Core Plus
1	Reset device	Core Plus

Reference : Ed. 5. Implementation Table, manufacturers’ meeting in Bologna 16/07/2003

6. Appendix 13 - Minimum Acceptable Implementations

From the wide range of ccTalk commands to choose from in Appendix 1, it can be hard for peripheral manufacturers to know which ones outside the 'Core Commands' need to be implemented. This then is **the requirement for a minimum acceptable implementation**. More commands may be needed and these are shown in separate sections. For instance, if the coin acceptor has a sorter then sorter commands should be added.

7.8 Coin Acceptors

254 Simple poll
246 Request manufacturer id
245 Request equipment category id
244 Request product code
242 Request serial number
241 Request software revision
232 Perform self-check
231 Modify inhibit status
230 Request inhibit status
229 Read buffered credit or error codes
228 Modify master inhibit status
227 Request master inhibit status
216 Request data storage availability
197 Calculate ROM checksum
192 Request build code
184 Request coin id
004 Request comms revision
001 Reset device

7.8.1 Add for Sorter / Diverter Support

222 Modify sorter override status
221 Request sorter override status
210 Modify sorter paths
209 Request sorter paths
189 Modify default sorter path
188 Request default sorter path

7.8.2 Add for Encrypted Serial Communication

137 Switch encryption code
136 Store encryption code
112 Read encrypted events
111 Request encryption support
110 Switch encryption key
108 Request encrypted monetary id

7.8.3 Add for Date Support

196 Request creation date
195 Request last modification date
170 Request base year

7.8.4 Add for Auditing

226 Request insertion counter
225 Request accept counter
194 Request reject counter
193 Request fraud counter

6.1 Payouts

254 Simple poll
246 Request manufacturer id
245 Request equipment category id
244 Request product code
242 Request serial number
241 Request software revision
217 Request payout high / low status
216 Request data storage availability
192 Request build code
172 Emergency stop
171 Request hopper coin
168 Request hopper dispense count
167 Dispense hopper coins
166 Request hopper status
164 Enable hopper
163 Test hopper
004 Request comms revision
001 Reset device

7.8.5 Add for Encrypted Payout

161 Pump RNG
160 Request cipher key

7.8.6 Add for Encrypted Serial Communication

137 Switch encryption code
136 Store encryption code
111 Request encryption support
110 Switch encryption key
109 Request encrypted hopper status

7.8.7 Add for Additional Security

219 Enter new PIN number
218 Enter PIN number
197 Calculate ROM checksum

7.8.8 Add for Date Support

196 Request creation date
195 Request last modification date
170 Request base year

7.8.9 Add for Parameter Support

247 Request variable set
165 Modify variable set

7.8.10 Add for Accumulator Mode

134 Dispense hopper value (instead of **167** Dispense hopper coins)
133 Request hopper polling value (instead of **166** Request hopper status)
132 Emergency stop value (instead of **172** Emergency stop)
131 Request hopper coin value (instead of **171** Request hopper coin)
130 Request indexed hopper dispense count (instead of **168** Request hopper dispense count)

6.2 Bill Validators

254 Simple poll
247 Request variable set
246 Request manufacturer id
245 Request equipment category id
244 Request product code
242 Request serial number
241 Request software revision
232 Perform self-check
231 Modify inhibit status
230 Request inhibit status
228 Modify master inhibit status
227 Request master inhibit status
216 Request data storage availability
213 Request option flags
197 Calculate ROM checksum
192 Request build code
159 Read buffered bill events
157 Request bill id
156 Request country scaling factor
154 Route bill
153 Modify bill operating mode
152 Request bill operating mode

145 Request currency revision
141 Request firmware upgrade capability
004 Request comms revision
001 Reset device

7.8.11 Add for Encrypted Serial Communication

137 Switch encryption code
136 Store encryption code
112 Read encrypted events
111 Request encryption support
110 Switch encryption key
108 Request encrypted monetary id

7.8.12 Add for Date Support

196 Request creation date
195 Request last modification date
170 Request base year

7.8.13 Add for Auditing

226 Request insertion counter
225 Request accept counter
194 Request reject counter
193 Request fraud counter
150 Request individual accept counter
149 Request individual error counter

7.8.14 Add for Barcode Support

129 Read barcode data

7.8.15 Add for Bank Switching Support

179 Modify bank select
178 Request bank select

Changers

254 Simple poll
247 Request variable set
246 Request manufacturer id
245 Request equipment category id
244 Request product code
242 Request serial number
241 Request software revision
232 Perform self-check
231 Modify inhibit status
230 Request inhibit status
216 Request data storage availability
197 Calculate ROM checksum
196 Request creation date
195 Request last modification date
192 Request build code
184 Request coin id
170 Request base year
141 Request firmware upgrade capability
128 Request money in
127 Request money out
126 Clear money counters
125 Pay money out
124 Verify money in
123 Request activity register
122 Request error status
121 Purge hopper
120 Modify hopper balance
119 Request hopper balance
118 Modify cashbox value
117 Request cashbox value
116 Modify real time clock
115 Request real time clock
004 Request comms revision
001 Reset device

8 Appendix 14 - Large Packet Exchange

The maximum ccTalk payload is 255 bytes of data. Since the 2nd byte of a ccTalk data packet is the data length then variable length messages can be sent very efficiently. A question often arises of how to send data blocks much larger than this. For instance, we may be trying to send new coin tables, bill tables or firmware into a device. Alternatively, we may be trying to read memory blocks or diagnostic reports from a device.

The recommended procedure is as follows…

8.1 Host to Device

Send a **Start** packet
Send multiple **Data** packets
(the last data packet may be less than the standard size)
Send a **Finish** packet

As the device knows in advance about data arriving and what it is for, the host does not need to send packet addresses or sequence numbers, although it may of course do so. It is assumed all data packets will arrive in sequence. Transfer integrity can be guaranteed by using large packet checksums at the end of the data.

As an example, refer to the bill table update procedure…

```
Start  = Header 143, Begin bill table upgrade
Data   = Header 144, Upload bill tables
Finish = Header 142, Finish bill table upgrade
```

This particular command transfers up to 128 bytes of data each time as shorter messages are better protected by the packet checksum and internal memory boundary calculations are often easier using powers of 2.

An error in data transfer will usually be picked up at the end when a final checksum is calculated. If an ACK is not received for one of the data packets then this packet could be re-sent but in this case a sequence number would be essential to prevent multiple copies being received in error.

8.2 Device to Host

Host requests **Memory Size** packet
Host polls **Addressed Data** packets

Since the host is the bus master it must first find out how much data the peripheral device has to send through a memory size packet request. The host can then poll the individual data packets out of the device but must include a block address.

As an example, refer to the user data commands…

```
Memory Size    = Header 216, Request data storage availability
Addressed Data = Header 215, Read data block
```

The data storage in this case can be anything from 1 to 65,280 bytes.

If an error occurs reading the data, then the request can be re-sent as each packet includes a block address.

Note that a more consistent solution is being developed for large packet transfers in new ccTalk applications in the year 2012 onwards; refer to command header 105, 'Data stream'.

9 Appendix 15 – Bill Types and Bill Values

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…

Bill validators return a country-specific scaling factor and decimal places – see command header 156, 'Request country scaling factor'.

The value code x scaling factor should express the bill value in terms of the lowest value currency unit in active circulation e.g. pence or cents.

Combined with the decimal places the result should be in terms of the bill currency unit e.g. pounds, euros or dollars.

Examples

= preferred settings

Value Code	Scaling Factor	Decimal Places	Result
0001	100	2	1.00
0002	100	2	2.00
0003	100	2	3.00
0005	100	2	5.00
0010	100	2	10.00
0020	100	2	20.00
0025	100	2	25.00
0030	100	2	30.00
0050	100	2	50.00
0100	100	2	100.00
0200	100	2	200.00
0250	100	2	250.00
0300	100	2	300.00
0500	100	2	500.00
1000	100	2	1,000.00
2000	100	2	2,000.00
2500	100	2	2,500.00
3000	100	2	3,000.00
5000	100	2	5,000.00
0001	10	2	0.10
0002	10	2	0.20
0003	10	2	0.30
0005	10	2	0.50
0010	10	2	1.00
0020	10	2	2.00
0025	10	2	2.50
0030	10	2	3.00
0050	10	2	5.00
0100	10	2	10.00

0200	10	2	20.00
0250	10	2	25.00
0300	10	2	30.00
0500	10	2	50.00
1000	10	2	100.00
2000	10	2	200.00
2500	10	2	250.00
3000	10	2	300.00
5000	10	2	500.00
0001	1,000	2	10.00
0002	1,000	2	20.00
0003	1,000	2	30.00
0005	1,000	2	50.00
0010	1,000	2	100.00
0020	1,000	2	200.00
0025	1,000	2	250.00
0030	1,000	2	300.00
0050	1,000	2	500.00
0100	1,000	2	1,000.00
0200	1,000	2	2,000.00
0250	1,000	2	2,500.00
0300	1,000	2	3,000.00
0500	1,000	2	5,000.00
1000	1,000	2	10,000.00
2000	1,000	2	20,000.00
2500	1,000	2	25,000.00
3000	1,000	2	30,000.00
5000	1,000	2	50,000.00
0001	10,000	2	100.00
0002	10,000	2	200.00
0003	10,000	2	300.00
0005	10,000	2	500.00
0010	10,000	2	1,000.00
0020	10,000	2	2,000.00
0025	10,000	2	2,500.00
0030	10,000	2	3,000.00
0050	10,000	2	5,000.00
0100	10,000	2	10,000.00
0200	10,000	2	20,000.00
0250	10,000	2	25,000.00
0300	10,000	2	30,000.00
0500	10,000	2	50,000.00
1000	10,000	2	100,000.00
2000	10,000	2	200,000.00
2500	10,000	2	250,000.00
3000	10,000	2	300,000.00
5000	10,000	2	500,000.00
Smallest likely (decimal places = 2)			
0001	1	2	0.01
Largest possible			
9999	65,535	0	655,284,465

Problems can exist in concurrent new & old currencies where there is a massive change in scaling factor as ccTalk only supports a common scaling factor for each country rather than a scaling factor for each note. For details of industry-wide solutions then contact Crane Payment Solutions - Money Controls for more information.

9.1 Unprogrammed Bills and Erased Bills

Any bill positions that have been left unprogrammed or have been subsequently erased from the bill validator should return a blank identifier. By convention this is 7 x ASCII dots (decimal code 46) rather than 7 x ASCII spaces (decimal code 32).

e.g.

```
Bill 1 = EU0020A
Bill 9 = .......
```

Bill 1 has a programmed banknote. Bill 9 is unprogrammed.

9.2 Use of Decimal Points in the Value Code

There are a few currencies where the ratio of highest denomination note to smallest denomination note exceeds the normal 1000:1 range.

For instance, in Brunei, they have 100 sens to 1 Brunei dollar (Malay ringgit) and a biggest note denomination of 10,000 dollars giving a highest to lowest monetary ratio of one million to 1 !

There are 2 solutions :-

a) Use a scaling factor of 100 and 2 decimal places with note codes from BN0001A to BN1000A. In this case there is no support for the highest value note (unlikely to be entered into an automatic cash handler).
b) Use a scaling factor of 1000 and 2 decimal places with notes codes from BN0.10A to BN1000A. This introduces a necessary decimal point into the value code but everything from a B$10,000 note to a B$0.01 coin is supported.

See RFC/002 which suggests the introduction of a decimal point into character positions 1, 2 or 3. Also see Appendix 3 which already allows for decimal points in coin codes to represent ‘half values’ and smaller currency units.

10 Appendix 16 - Bill Acceptor Messaging Example

6.3 Initialisation

This is a typical initialisation or enrolment process for a gaming machine. The idea is to confirm the ccTalk link to the coin acceptor is operational and that the device fitted is approved for use in this environment.

The example here is shown without encryption and using 8-bit checksums.

Simple poll = ACK (confirms a ccTalk device is operating at address 40)
```
TX = 040 000 001 254 217
RX = 001 000 040 000 215
```

Request equipment category id = ‘Bill Acceptor’
```
TX = 040 000 001 245 226
RX = 001 013 040 000 066 105 108 108 032 065 099 099 101 112 116 111 114 246
```

Request product code = ‘Ardac6’
```
TX = 040 000 001 244 227
RX = 001 006 040 000 065 114 100 097 099 054 192
```

Request build code = ‘Standard’
```
TX = 040 000 001 192 023
RX = 001 008 040 000 083 116 097 110 100 097 114 100 158
```

Request manufacturer id = ‘MCI’
```
TX = 040 000 001 246 225
RX = 001 003 040 000 077 067 073 251
```

Request serial number = 1605
```
TX = 040 000 001 242 229
RX = 001 003 040 000 069 006 000 137
```

Request software revision = ‘AEN-F1-V4.44’
```
TX = 040 000 001 241 230
RX = 001 012 040 000 065 069 078 045 070 049 045 086 052 046 052 052 006
```

Request comms revision = 1.4.4
```
TX = 040 000 001 004 211
RX = 001 003 040 000 001 004 004 203
```

Calculate checksum = **0x3A43045F**
```
TX = 040 000 001 197 018
RX = 001 004 040 000 095 004 067 058 243
```

The checksum calculation may take a few seconds to reply – refer to the product manual for more information.

6.4 Pre-Acceptance

Request bill id for bill 1 = 'EU0005A'
```
TX = 040 001 001 157 001 056
RX = 001 007 040 000 069 085 048 048 048 053 065 048
```

Request bill id for bill 2 = 'EU0010A'
```
TX = 040 001 001 157 002 055
RX = 001 007 040 000 069 085 048 048 049 048 065 052
```

Request bill id for bill 3 = 'EU0020A'
```
TX = 040 001 001 157 003 054
RX = 001 007 040 000 069 085 048 048 050 048 065 051
```

Request bill id for bill 4 = 'EU0050A'
```
TX = 040 001 001 157 004 053
RX = 001 007 040 000 069 085 048 048 053 048 065 048
```

Request bill id for bill 5 = 'EU0100A'
```
TX = 040 001 001 157 005 052
RX = 001 007 040 000 069 085 048 049 048 048 065 052
```

Request bill id for bill 6 = 'EU0200A'
```
TX = 040 001 001 157 006 051
RX = 001 007 040 000 069 085 048 050 048 048 065 051
```

Request bill id for bill 7 = 'EU0500A'
```
TX = 040 001 001 157 007 050
RX = 001 007 040 000 069 085 048 053 048 048 065 048
```

Request bill id for bill 8 = '……..' (not programmed)
```
TX = 040 001 001 157 008 049
RX = 001 007 040 000 046 046 046 046 046 046 046 142
```

Request country scaling factor for EU = x 100 with 2 decimal places
```
TX = 040 002 001 156 069 085 159
RX = 001 003 040 000 100 000 002 110
```

Request currency revision for EU = 001
```
TX = 040 002 001 145 069 085 170
RX = 001 003 040 000 048 048 049 067
```

Modify bill operating mode = ACK (stacker enabled, escrow enabled)
```
TX = 040 001 001 153 003 058
RX = 001 000 040 000 215
```

Modify inhibit status = ACK
```
TX = 040 008 001 231 255 255 255 255 255 255 255 255 240
RX = 001 000 040 000 215
```

Modify master inhibit status = ACK
```
TX = 040 001 001 228 001 241
RX = 001 000 040 000 215
```

All bills are now enabled for acceptance. Credit polling should commence within 5s else the bill validator will self-inhibit.

6.5 Credit Polling

Poll every 200ms and don't stop…

10.1.1 Example 1 – Accepted & Rejected Notes

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 001 003 001 000 000 000 000 000 000 000 000 199

Event code has changed from 0 to 1 : Bill 3 held in escrow
```

Route bill to stacker / cashbox = ACK
```
TX = 040 001 001 154 001 059
RX = 001 000 040 000 215
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 002 003 000 003 001 000 000 000 000 000 000 195

Event code has changed from 1 to 2 : Bill 3 stacked OK
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 004 000 001 000 002 003 000 003 001 000 000 190

Event code has changed from 2 to 4 : Invalid bill ( due to validation fail )
followed by Bill returned from escrow
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 006 000 001 000 003 000 001 000 002 003 000 188

Event code has changed from 4 to 6 : Invalid bill ( due to transport problem
) followed by Bill returned from escrow
```

10.1.2 Example 2 – Coupon Acceptance

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 002 255 001 000 020 000 000 000 000 000 000 182
```

```
Event code has changed from 0 to 2 : Barcode detected followed by coupon
held in escrow. A coupon has a nominal credit code of 255.
```

Read barcode data
```
TX = 040 000 001 129 086
RX = 001 018 040 000 052 055 053 057 054 050 053 055 053 053 056 055 055 049
048 050 051 049 017
```

```
Barcode = 475962575587710231
```

Route bill to stacker / cashbox = ACK
```
TX = 040 001 001 154 001 059
RX = 001 000 040 000 215
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 003 255 000 255 001 000 020 000 000 000 000 182
```

```
Event code has changed from 2 to 3 : Coupon stacked OK
```

Other examples…

Non-escrow mode / Barcode accepted

```
Buffered bills = 02 FF 00 00 14 00 00 00 00 00 00
Event Counter = 2
The last 5 events were...
1. Credit 255 Accepted
2. Barcode detected ( host to use header 129 / Read barcode data )
3. Master inhibit active
4. Master inhibit active
5. Master inhibit active
```

Non-escrow mode / Barcode rejected

```
Buffered bills = 01 00 02 00 00 00 00 00 00 00 00
Event Counter = 1
The last 5 events were...
1. Invalid bill ( due to validation fail )
2. Master inhibit active
3. Master inhibit active
4. Master inhibit active
5. Master inhibit active
```

Escrow mode / Barcode accepted

```
Buffered bills = 03 FF 00 FF 01 00 14 00 00 00 00
Event Counter = 3
The last 5 events were...
1. Credit 255 Accepted
2. Credit 255 Escrow ( host to use header 154 / Route bill )
3. Barcode detected ( host to use header 129 / Read barcode data )
4. Master inhibit active
5. Master inhibit active
```

Escrow mode / Barcode rejected

```
Buffered bills = 01 00 02 00 00 00 00 00 00 00 00
Event Counter = 1
The last 5 events were...
1. Invalid bill ( due to validation fail )
2. Master inhibit active
3. Master inhibit active
4. Master inhibit active
5. Master inhibit active
```

10.1.3 Example 3 – Stringing Attack

Read buffered bill events

```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events

```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events

```
TX = 040 000 001 159 056
RX = 001 011 040 000 002 000 001 000 009 000 000 000 000 000 000 192
```

```
Event code has changed from 0 to 2 : Bill tamper followed by Bill returned
from escrow
```

10.1.4 Example 4 – Polling Timeout & Recovery

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 002 000 001 000 002 000 000 000 000 000 000 199
```

```
Event code has changed from 0 to 2 : Invalid bill ( due to validation fail )
followed by Bill returned from escrow
```

Host stops polling for more than 5s and then resumes…

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 003 000 000 000 001 000 002 000 000 000 000 198
```

```
Event code has changed from 2 to 3 : Master inhibit active
```

Modify master inhibit status = ACK
```
TX = 040 001 001 228 001 241
RX = 001 000 040 000 215
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 003 000 000 000 001 000 002 000 000 000 000 198
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 003 000 000 000 001 000 002 000 000 000 000 198
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 004 000 002 000 000 000 001 000 002 000 000 195
```

```
Event code has changed from 3 to 4 : Invalid bill ( due to validation fail )
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 005 000 001 000 002 000 000 000 001 000 002 193
```

```
Event code has changed from 4 to 5 : Bill returned from escrow
```

10.1.5 Example 5 – Stacker Full Message

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 000 000 000 000 000 000 000 000 000 000 000 204
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 002 255 001 000 020 000 000 000 000 000 000 182
```

```
Event code has changed from 0 to 2 : Barcode detected followed by coupon
held in escrow. A coupon has a nominal credit code of 255.
```

Read barcode data
```
TX = 040 000 001 129 086
RX = 001 018 040 000 052 055 053 057 054 050 053 055 053 053 056 055 055 049
048 050 051 049 017
```

```
Barcode = 4759625755877102 31
```

Route bill to stacker / cashbox = ACK
```
TX = 040 001 001 154 001 059
RX = 001 000 040 000 215
```

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 003 255 000 255 001 000 020 000 000 000 000 182
```

```
Event code has changed from 2 to 3 : Coupon stacked OK
```

Stacker is now full

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 005 000 000 000 014 255 000 255 001 000 020 166
```

```
Event code has changed from 3 to 5 : Stacker full followed by Master inhibit
active
```

Bill validator self inhibits to prevent additional notes being inserted

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 006 000 011 000 000 000 014 255 000 255 001 174
```

```
Event code has changed from 5 to 6 : Stacker removed
```

Stacker is emptied

Read buffered bill events
```
TX = 040 000 001 159 056
RX = 001 011 040 000 007 000 012 000 011 000 000 000 014 255 000 161
```

```
Event code has changed from 6 to 7 : Stacker inserted
```

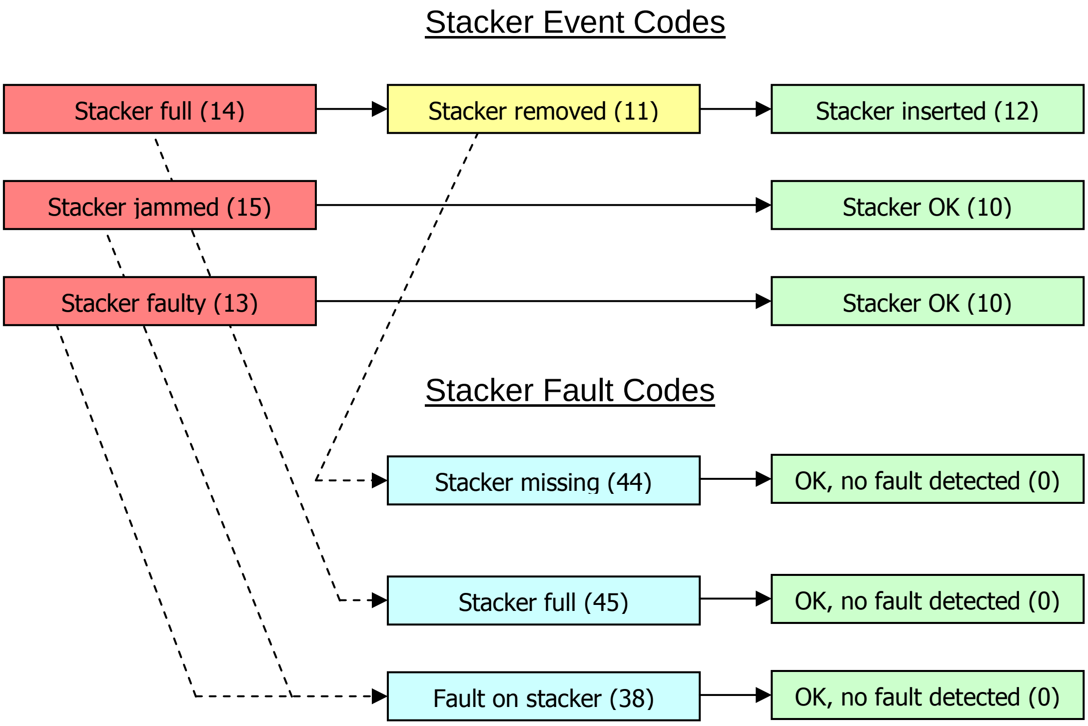
Modify master inhibit status = ACK
```
TX = 040 001 001 228 001 241
RX = 001 000 040 000 215
```

Bill validator is now ready to accept and stack more coupons or notes

Remember that the event code wraps from 255 to 1 as 0 is a special start-up value indicating a power-up or reset has occurred.

6.6 Stacker Events & Faults

This schematic should clarify the relationship between the different stacker event and fault codes.



11 Table 1 - ccTalk Standard Category Strings & Default Addresses

The following standard category strings have been defined, along with their typical default addresses. Extra addresses may be used where more than 1 type of device category exists on the same bus. Note that the ccTalk protocol allows the addresses to be changed to arbitrary values in multi-drop applications.

These standard category strings should be reported exactly as reproduced in the table below by the 'Read equipment category id' command. Observe any capitalisation rules - 'Coin Acceptor' is not the same as 'COIN ACCEPTOR', 'coin acceptor' or 'Coin acceptor'.

The ccTalk standard category is sometimes referred to as the ccTalk 'class'.

Standard Category	Address	Extra Addresses	Comments…
Coin Acceptor	2	11 to 17	a.k.a Coin Validator
Payout	3	4 to 10	a.k.a Hopper
Reel	30	31 to 34	
Bill Validator	40	41 to 47	a.k.a Note Acceptor
Card Reader	50		
Changer	55		Money-in, money-out recyclers. Also used for coin singulators and sorters.
Display	60		e.g. LCD panels, alpha-numeric displays
Keypad	70		Remote keyboard
Dongle	80	85 to 89	Security device, interface box or interface hub
Meter	90		Electro-mechanical counter replacement
Bootloader	99		Bootloader firmware and diagnostics when no application code is loaded.
Power	100		Power switching hub or intelligent power supply
Printer	110		Ticket printer for coupons and barcodes
RNG	120		Random Number Generator
Hopper Scale	130		Hopper with weigh scale
Coin Feeder	140		Motorised coin feeder or singulator
Bill Recycler	150		Bill or note recyclers
Escrow	160		Motorised escrow
Debug	240	241 to 255	This address range may be used when developing new peripherals

12 Table 2 - ccTalk Coin Acceptor Error Code Table

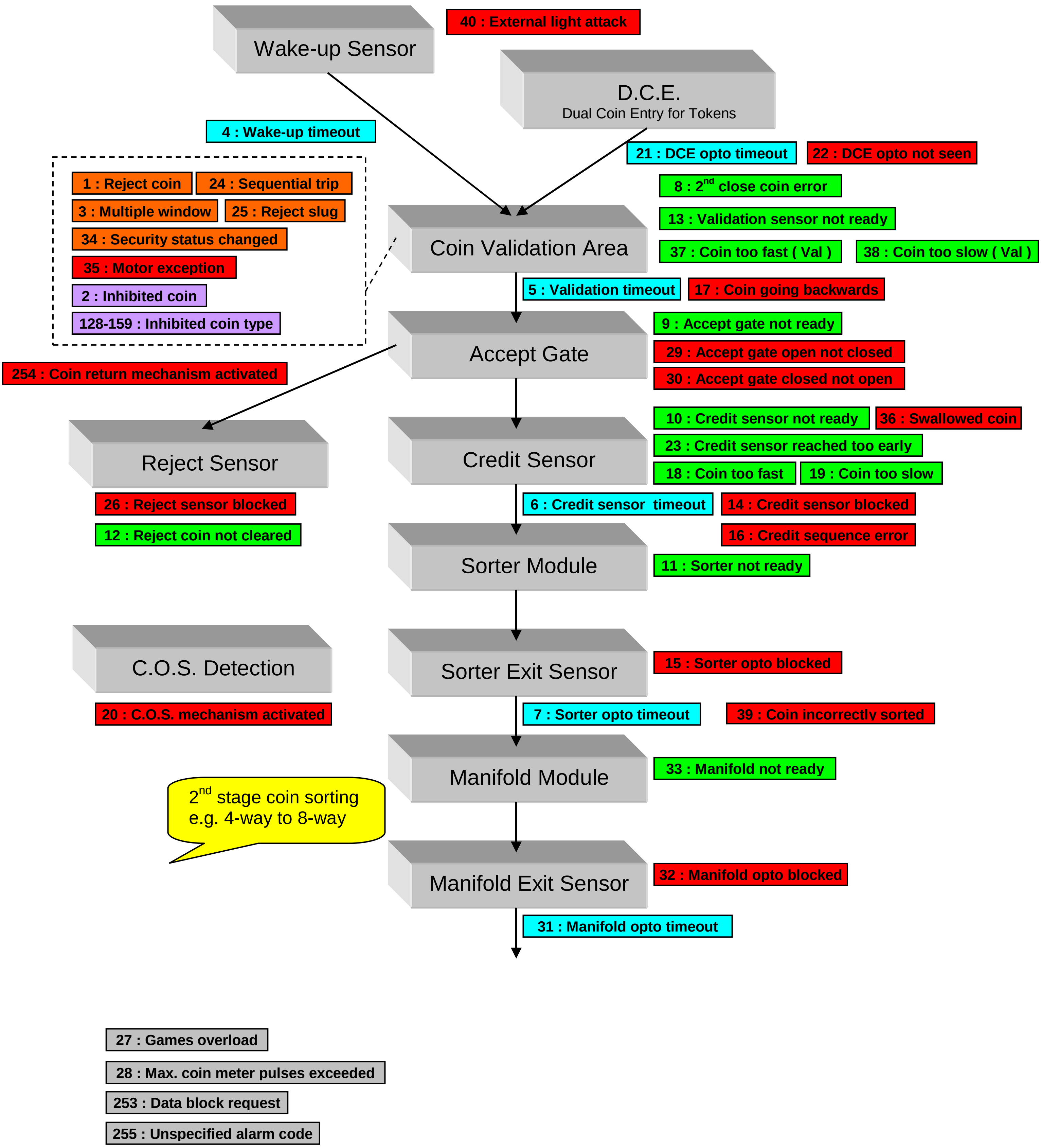
The following standard errors have been defined for a coin acceptor. Not all may be implemented so refer to the relevant product manual. A serial credit code indicates that a coin was definitely accepted. A serial error code may or may not indicate a coin was rejected due to most coin acceptors not having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

Code	Error	Coin rejected ?
0	Null event (no error)	No
1	Reject coin	**Yes** - by definition
2	Inhibited coin	**Yes**
3	Multiple window	**Yes**
4	Wake-up timeout	Possible
5	Validation timeout	Possible
6	Credit sensor timeout	Possible
7	Sorter opto timeout	No
8	2nd close coin error	**Yes** - 1 or more
9	Accept gate not ready	**Yes**
10	Credit sensor not ready	**Yes**
11	Sorter not ready	**Yes**
12	Reject coin not cleared	**Yes**
13	Validation sensor not ready	**Yes**
14	Credit sensor blocked	**Yes**
15	Sorter opto blocked	**Yes**
16	Credit sequence error	No
17	Coin going backwards	No
18	Coin too fast (over credit sensor)	No
19	Coin too slow (over credit sensor)	No
20	C.O.S. mechanism activated (coin-on-string)	No
21	DCE opto timeout	Possible
22	DCE opto not seen	**Yes**
23	Credit sensor reached too early	No
24	Reject coin (repeated sequential trip)	**Yes**
25	Reject slug	**Yes**
26	Reject sensor blocked	No
27	Games overload	No
28	Max. coin meter pulses exceeded	No
29	Accept gate open not closed	No
30	Accept gate closed not open	**Yes**
31	Manifold opto timeout	No
32	Manifold opto blocked	**Yes**
33	Manifold not ready	**Yes**
34	Security status changed	Possible
35	Motor exception	Possible
36	Swallowed coin	No

37	Coin too fast (over validation sensor)	**Yes**
38	Coin too slow (over validation sensor)	**Yes**
39	Coin incorrectly sorted	No
40	External light attack	No
128	Inhibited coin (Type 1)	**Yes**
…	Inhibited coin (Type n)	**Yes**
159	Inhibited coin (Type 32)	**Yes**
160	*Reserved (credit cancelling mechanism)*	**-**
…	*Reserved*	**-**
191	*Reserved*	**-**
253	Data block request (note α)	No
254	Coin return mechanism activated (flight deck open)	No
255	Unspecified alarm code	No

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

12.1 Pictorial View of Coin Acceptor Error / Event Codes



12.2 ccTalk Coin Acceptor Error Code Descriptions

These codes are specific to coin acceptors. Bill validators and payout devices use their own error code tables. A manufacturer of ccTalk coin acceptors can implement any subset of the error codes below depending on the technology they are using and the ability to self-diagnose problems. These are all ‘event’ codes and can be reported at any time in a reply to a host credit poll.

Code	Error	**Description**
1	Reject coin	**A coin was inserted which did not match any of the programmed types. The coin is returned to the customer and no credit is given.**
2	Inhibited coin	**A coin was inserted which did match a programmed window type but was prevented from accepting by the inhibit register. The inhibit register can be controlled serially but may also be linked to external DIL switches.**
3	Multiple window	**A coin was inserted which matched more than one enabled window type. This coin was rejected as the credit code was indeterminate.**
4	Wake-up timeout	**A coin acceptor fitted with a wake-up sensor picked up a coin entering the acceptor but it was not seen subsequently in the validation area. Possible coin jam.**
5	Validation timeout	**A coin was detected entering the validation area but failed to leave it. Possible coin jam.**
6	Credit sensor timeout	**A coin was validated as true but never made it to the post-gate credit sensor. Possible coin jam.**
7	Sorter opto timeout	**A coin was sent into the sorter / diverter but was not seen coming out. Possible coin jam.**
8	2nd close coin error	**A coin was inserted too close to the one in front. One or both coins will have rejected.**
9	Accept gate not ready	**A coin was inserted while the accept gate for the coin in front was still operating. Coins have been inserted too quickly.**
10	Credit sensor not ready	**A coin was still over the credit sensor when another coin was ready to accept. Coins have been inserted too quickly.**
11	Sorter not ready	**A coin was inserted while the sorter flaps for the coin in front were still operating. Coins have been inserted too quickly.**
12	Reject coin not cleared	**A coin was inserted before a previously rejected coin had time to clear the coin acceptor. Coins have been inserted too quickly.**
13	Validation sensor not ready	**The validator inductive sensors were not ready for coin validation. Possible fault developing.**
14	Credit sensor blocked	**There is a permanent blockage at the credit sensor. The coin acceptor will not accept any more coins.**
15	Sorter opto blocked	**There is a permanent blockage at the sorter exit sensor. The coin acceptor will not accept any more coins.**
16	Credit sequence error	**A coin or object was detected going backwards through a directional credit sensor. Possible fraud attempt.**
17	Coin going backwards	**A coin was detected going backwards through the coin acceptor. Possible fraud attempt.**
18	Coin too fast (over credit sensor)	**A coin was timed going through the credit sensor and was too fast. Possible fraud attempt.**
19	Coin too slow (over credit sensor)	**A coin was timed going through the credit sensor and was too slow. Possible fraud attempt.**
20	C.O.S. mechanism activated (coin-on-string)	**A specific sensor for detecting a ‘coin on string’ was activated. Possible fraud attempt.**

21	DCE opto timeout	**A coin acceptor fitted with a Dual Coin Entry chute saw a coin or token which was not seen subsequently in the validation area. Possible coin jam.**
22	DCE opto not seen	**A coin acceptor fitted with a Dual Coin Entry chute saw a coin which was not seen previously by the chute sensor. Possible fraud attempt.**
23	Credit sensor reached too early	**A coin was timed from the end of the validation area to the post-gate credit sensor. It arrived too early. Possible fraud attempt.**
24	Reject coin (repeated sequential trip)	**A coin was rejected N times in succession with no intervening true coins. Statistically unlikely if N greater than or equal to 5. Possible fraud attempt.**
25	Reject slug	**A coin was rejected but was identified as a known slug type - this may be a pre-programmed fraud coin or a known fraud material.**
26	Reject sensor blocked	**There is a permanent blockage at the reject sensor. The coin acceptor will not accept any more coins. Not all coin acceptors have a reject sensor.**
27	Games overload	**Totaliser mode : A game value was set too low - possibly zero. This is a product configuration error.**
28	Max. coin meter pulses exceeded	**Totaliser mode : A meter value was set too low - possibly zero. This is a product configuration error.**
29	Accept gate open not closed	**The accept gate was forced open when it should have been closed.**
30	Accept gate closed not open	**The accept gate did not open when the solenoid was driven.**
31	Manifold opto timeout	**A coin was sent into the manifold module (coin diverter) but was not seen coming out. Possible coin jam.**
32	Manifold opto blocked	**There is a permanent blockage at the manifold module sensor (coin diverter). The coin acceptor will not accept any more coins.**
33	Manifold not ready	**A coin was inserted while the manifold flap for the coin in front was still operating. Coins have been inserted too quickly.**
34	Security status changed	**The coin acceptor changed its security status (coin acceptance criteria) based on the detection of fraudulent activity. Refer to ccTalk header 180.**
35	Motor exception	**For coin acceptors using a motor, this indicates some kind of motor problem such as a coin jam.**
36	Swallowed coin	**A coin was seen at the credit sensor when it should have rejected. The coin value is unknown. There may have been a fraudulent manipulation of the accept gate or a hardware failure (gate stuck open).**
37	Coin too fast (over validation sensor)	**A coin was timed going through the validation sensor area and was too fast. The coin may have been 'flicked' in.**
38	Coin too slow (over validation sensor)	**A coin was timed going through the validation sensor area and was too slow. The coin may be sticking or is being held on a string (fraud attempt).**
39	Coin incorrectly sorted	**A coin was accepted but sent to the wrong sorter path. It may be indicative of a hardware fault. This is just a notification event to say it happened without further details.**
40	External light attack	**A sensor detected external light. This may be an attempt to fraud the opto sensors.**
128	Inhibited coin (Type 1)	**A true coin (type 1, coin in position 1) was inserted but was prevented from accepting by the inhibit register.**

…	Inhibited coin (Type n)	**A true coin (type n, coin in position n) was inserted but was prevented from accepting by the inhibit register.**
159	Inhibited coin (Type 32)	**A true coin (type 32, coin in position 32) was inserted but was prevented from accepting by the inhibit register.**
253	Data block request (note α)	**A 'not yet used' mechanism for a coin acceptor to request attention from the host machine. Perhaps it needs some data from the host machine or another peripheral.**
254	Coin return mechanism activated (flight deck open)	**An attempt to clear a coin jam by opening the flight deck was detected. The coin acceptor cannot operate until the flight deck is closed.**
255	Unspecified alarm code	**Any alarm code which does not fit into the above categories.**

13 Table 3 - ccTalk Fault Code Table

Code	Fault	Optional Extra Info
0	OK (no fault detected)	-
1	EEPROM checksum corrupted	-
2	Fault on inductive coils	Coil number
3	Fault on credit sensor	-
4	Fault on piezo sensor	-
5	Fault on reflective sensor	-
6	Fault on diameter sensor	-
7	Fault on wake-up sensor	-
8	Fault on sorter exit sensors	Sensor number
9	NVRAM checksum corrupted	-
10	Coin dispensing error	-
11	Low level sensor error	Hopper or tube number
12	High level sensor error	Hopper or tube number
13	Coin counting error	-
14	Keypad error	Key number
15	Button error	-
16	Display error	-
17	Coin auditing error	-
18	Fault on reject sensor	-
19	Fault on coin return mechanism	-
20	Fault on C.O.S. mechanism	-
21	Fault on rim sensor	-
22	Fault on thermistor	-
23	Payout motor fault	Hopper number
24	Payout timeout	Hopper or tube number
25	Payout jammed	Hopper or tube number
26	Payout sensor fault	Hopper or tube number
27	Level sensor error	Hopper or tube number
28	Personality module not fitted	-
29	Personality checksum corrupted	-
30	ROM checksum mismatch	-
31	Missing slave device	Slave address
32	Internal comms bad	Slave address
33	Supply voltage outside operating limits	-
34	Temperature outside operating limits	-
35	D.C.E. fault	1 = coin, 2 = token
36	Fault on bill validation sensor	Sensor number
37	Fault on bill transport motor	-
38	Fault on stacker	-
39	Bill jammed	-
40	RAM test fail	-
41	Fault on string sensor	-
42	Accept gate failed open	-

43	Accept gate failed closed	-
44	Stacker missing	-
45	Stacker full	-
46	Flash memory erase fail	-
47	Flash memory write fail	-
48	Slave device not responding	Device number
49	Fault on opto sensor	Opto number
50	Battery fault	-
51	Door open	-
52	Microswitch fault	-
53	RTC fault	-
54	Firmware error	-
55	Initialisation error	-
56	Supply current outside operating limits	-
57	Forced bootloader mode	-
255	Unspecified fault code	Further information

Any fault code can return optional information in the 2nd byte. Consult the product manual for more details.

14 ccTalk Fault Code Descriptions

This is a generic fault code table for all ccTalk devices.

A manufacturer of ccTalk equipment can implement any subset of the fault codes below depending on the technology they are using and the ability to self-diagnose problems. These are all status codes which can be returned in response to a 'Perform self-check' command. They are all 'fatal' errors in that any non-zero reply prevents normal operation of the device and is an implicit request to the host machine for a service callout. The host does not need to 'inhibit' or prevent the device from operating if a non-zero fault code is returned as this will be done automatically.

Note that all fault code conditions relating to payout devices were incorporated into the 'Test hopper' command in a past revision of the protocol and are therefore marked as obsolete.

The ccTalk protocol currently only supports the reporting of one fault code at a time and this may or may not be in priority order depending on the firmware complexity. If a device has multiple faults then the second one will be reported when the first one is fixed etc.

Code	Fault	**Description**
0	OK (no fault detected)	**The usual response. Everything is working.**
1	EEPROM checksum corrupted	**The coin acceptor has found a mismatch between the checksum calculated from a coin data area and a stored checksum. Possible EEPROM corruption. This checksum is not intended for use with program code / firmware.**
2	Fault on inductive coils	**A fault was detected with the coils for inductive coin validation.**
3	Fault on credit sensor	**A fault was detected with the post-gate credit sensor. A serial credit can only be generated if the coin passes this sensor.**
4	Fault on piezo sensor	**A fault was detected on the piezo sensor used for slug rejection.**
5	Fault on reflective sensor	**A fault was detected on an opto-reflective sensor for coin validation.**
6	Fault on diameter sensor	**A fault was detected on a validation sensor specifically reserved for diameter resolution.**
7	Fault on wake-up sensor	**A fault was detected on the sensor used to wake-up a coin acceptor from a sleeping or power-down state.**
8	Fault on sorter exit sensors	**A fault was detected on the sorter exit sensors. These sensors confirm that a coin has cleared the sorter flaps and perhaps to verify the path taken.**
9	NVRAM checksum corrupted	**If battery-backed RAM is used then a corrupted checksum was discovered.**
10	Coin dispensing error	**Obsolete : A fault was found during a hopper coin dispense operation.**
11	Low level sensor error	**Obsolete : A fault was found on a hopper low level sensor.**
12	High level sensor error	**Obsolete : A fault was found on a hopper high level sensor.**
13	Coin counting error	**Obsolete : A fault was detected in the hopper 'dead reckoning' system. It probably ran out of coins.**
14	Keypad error	**A fault was found on a keypad.**
15	Button error	**A fault was found on a button.**
16	Display error	**A fault was found on a display device.**
17	Coin auditing error	**A fault was detected in the memory block used to record the number of inserted and accepted coins on a coin acceptor.**
18	Fault on reject sensor	**A fault was detected with the reject sensor. This is the sensor used to confirm a coin has left the reject path and has been returned to the customer.**
19	Fault on coin return mechanism	**A fault was detected in the flight deck mechanism used by the customer to clear coin jams in the entry or validation area.**
20	Fault on C.O.S. mechanism	**A fault was found on the 'Coin on String' sensor.**
21	Fault on rim sensor	**A fault was found on a coin rim validation sensor.**

22	Fault on thermistor	**A fault was found on a thermistor used to measure ambient temperature.**
23	Payout motor fault	**A fault was found on a hopper motor (used on changers)**
24	Payout timeout	**Obsolete : A coin was dispensed from a hopper but was not seen on the payout verification sensor.**
25	Payout jammed	**Obsolete : A jam was detected in a hopper.**
26	Payout sensor fault	**A fault was found on a hopper payout verification sensor (used on changers)**
27	Level sensor error	**A fault was found on a hopper level sensor.**
28	Personality module not fitted	**A personality or configuration module needed with some ccTalk peripherals was not fitted.**
29	Personality checksum corrupted	**A data checksum on a personality or configuration module was corrupted.**
30	ROM checksum mismatch	**The device has found a mismatch between the checksum calculated from a program code area and a stored checksum. Possible flash memory / ROM corruption. Now used for any kind of application firmware checksum error.**
31	Missing slave device	**Obsolete : A ccTalk peripheral did not find an attached slave device. Only of use in multi-master systems.**
32	Internal comms bad	**A ccTalk peripheral could not access an internal serial device.**
33	Supply voltage outside operating limits	**The ccTalk device is operating outside supply voltage limits defined in the product specification.**
34	Temperature outside operating limits	**The ccTalk device is operating outside temperature limits defined in the product specification.**
35	D.C.E. fault	**A fault was found on the Dual Coin Entry chute.**
36	Fault on bill validation sensor	**A fault was found on one of the bill validator validation sensors.**
37	Fault on bill transport motor	**A fault was found on the motor used to drive a bill through the bill validator.**
38	Fault on stacker	**A fault was found on the stacker attached to a bill validator.**
39	Bill jammed	**A bill is stuck in the bill validator.**
40	RAM test fail	**A read / write test cycle of the bill validator RAM has indicated a fault.**
41	Fault on string sensor	**A fault was found on a sensor used for detecting bills on a string.**
42	Accept gate failed open	**The coin accept gate is stuck open due to a jam or fraud attempt.**
43	Accept gate failed closed	**The coin accept gate is stuck closed. Possible open-circuit fault on the solenoid driver.**
44	Stacker missing	**The stacker is not fitted and needs to be for notes to accept.**
45	Stacker full	**The stacker is full and needs emptying.**
46	Flash memory erase fail	**The last flash memory erase cycle did not complete successfully.**
47	Flash memory write fail	**The last flash memory write cycle did not complete successfully.**
48	Slave device not responding	**If an attached device acts as a host to other slave devices then this fault code indicates a failure to communicate with those other devices.**
49	Fault on opto sensor	**A fault was detected on an opto-electronic sensor.**
50	Battery fault	**A system battery is missing or low on charge and requires replacing / recharging.**
51	Door open	**A door on the system was left in the open position. It must be shut to continue.**
52	Microswitch fault	**A fault was detected on a microswitch.**
53	RTC fault	**A fault was detected on the Real Time Clock.**
54	Firmware error	**A non-checksum type error was detected in the firmware or the firmware of an attached peripheral.** *Self-calculated checksum errors should be reported in fault code 30.*
55	Initialisation error	**An initialisation error was detected in the peripheral on power-up. The optional extra info field can break this down further if required.**

56	Supply current outside operating limits	**The ccTalk device is operating outside supply current limits defined in the product specification. There may be a short or a faulty component.**
57	Forced bootloader mode	**An external input was used to force the device into a bootloader mode. From here firmware can be re-programmed and the device reset. This is a special mode which stops the main application firmware from running.**
255	Unspecified fault code	**Any fault code which does not fall into the above categories. Some manufacturers may wish to use the optional byte to specify a manufacturer-specific fault code.**

15 Table 4 - ccTalk Coin Acceptor Status Codes

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

16 Table 5 - ccTalk Coin Calibration Reply Codes

See obsolete header 200, ‘Upload coin data’ in part 4 of the specification.

Code	Error
1	calibration denied
2	calibration recharge required
3	calibration failed (product name mismatch)
4	calibration failed (database number mismatch)
250	calibration error (key not supported)
251	calibration error (internal bin failure)
252	calibration error (op-code not supported)
253	calibration error (illegal parameter)
254	calibration error (database corrupt)
255	calibration error (unspecified)

17 Table 6 - ccTalk Standard Manufacturer Strings

The following standard manufacturer strings have been registered by the ccTalk User Group. They are returned by the 'Request manufacturer id' command and can be used to help identify a specific product.

BNV's are expected to reply with abbreviated names. Other peripherals may return a full name.

17.1 Manufacturer Names

Full Name	Abbreviated Name
Aardvark Embedded Solutions Ltd	**AES**
Alberici	**ALB**
AlfaNet informatika d.o.o	**ANI**
AstroSystems Ltd	**AST**
Azkoyen	**AZK**
Comestero Group	**CMG**
Crane CashCode Company	**CCC**
Crane Payment Solutions	**CPS**
Encopim SL	**ECP**
Gaming Technology Distribution	**GTD**
Himecs	**HIM**
Industrias Lorenzo	**IL**
Innovative Technology Ltd	**ITL**
Intergrated*(sic)* Technology Ltd	**INT**
International Currency Technologies	**ICT**
Japan Cash Machine	**JCM**
Jofemar	**JOF**
Kuky	**KUK**
Mars Electronics International	**MEI**
Microsystem Controls Pty. Ltd. *(Microcoin)*	**MSC**
Money Controls *(International)*	**MCI**
National Rejectors Inc	**NRI**
Phoenix Mecano Digital	**PMD**
Starpoint Electrics Ltd	**SEL**
Telequip / Crane	**TQP**
Weavefuture Inc	**WFT**
WH Münzprüfer	**WHM**

(Italicised text in parentheses) is for comment and does not actually form part of the return string.

If you are a manufacturer of ccTalk peripherals then you may submit your company identification string for inclusion in this table.

18 Table 7 - ccTalk Bill Event Codes

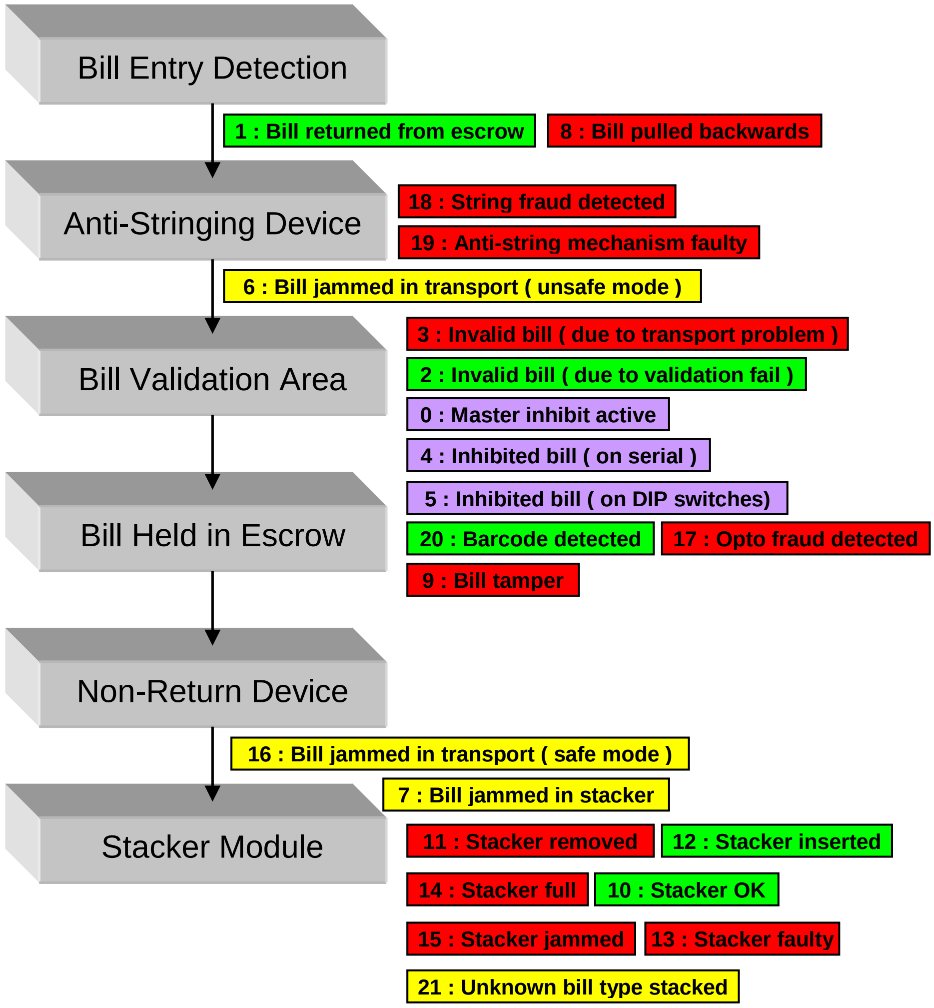
Result A	Result B	Event	Type
1 to 255	0	Bill type 1 to 255 validated correctly and sent to cashbox / stacker	Credit
1 to 255	1	Bill type 1 to 255 validated correctly and held in escrow	Pending Credit
0	0	Master inhibit active	Status
0	1	Bill returned from escrow	Status
0	2	Invalid bill (due to validation fail)	Reject
0	3	Invalid bill (due to transport problem)	Reject
0	4	Inhibited bill (on serial)	Status
0	5	Inhibited bill (on DIP switches)	Status
0	6	Bill jammed in transport (unsafe mode)	Fatal Error
0	7	Bill jammed in stacker	Fatal Error
0	8	Bill pulled backwards	Fraud Attempt
0	9	Bill tamper	Fraud Attempt
0	10	Stacker OK	Status
0	11	Stacker removed	Status
0	12	Stacker inserted	Status
0	13	Stacker faulty	Fatal Error
0	14	Stacker full	Status
0	15	Stacker jammed	Fatal Error
0	16	Bill jammed in transport (safe mode)	Fatal Error
0	17	Opto fraud detected	Fraud Attempt
0	18	String fraud detected	Fraud Attempt
0	19	Anti-string mechanism faulty	Fatal Error
0	20	Barcode detected	Status
0	21	Unknown bill type stacked	Status

There are two types of ‘Bill jammed in transport’ errors - safe mode and unsafe mode. The safe mode assumes that the note is jammed in a position which cannot be retrieved by the customer and so if validated as true a credit can be given. The unsafe mode assumes that the customer can retrieve the note and so no credit should be given.

Event Types

Credit	Bill accepted - credit the customer
Pending Credit	Bill held in escrow - decide whether to accept it
Reject	Bill rejected and returned to customer
Fraud Attempt	Fraud detected. Possible machine alarm.
Fatal Error	Service callout
Status	Informational only

18.1 Pictorial View of Bill Validator Event Codes



19 Table 8 – ccTalk Packet Lengths

The expected number of ccTalk data bytes are shown in the table below. This is the value in the 2nd byte of each ccTalk packet. Min / max values may differ when there are options within that command for the formatting of data. Sometimes there are preferred values within this range. Any peripherals which fall outside the range shown do not comply with the ccTalk specification.

When the TX values are zero, there is no transmit data which is sent with the ccTalk header in each packet. When the RX values are zero the reply is a simple ACK.

For example header 241, ‘Request software revision’, has no transmit data but the receive data string must contain between 3 and 32 characters, with no preferred value. This is similar to header 246, ‘Request manufacturer id’, but in this case there is a preferred return string length of 3 to be compatible with the abbreviated name format in Table 6 of the ccTalk specification.

Header	Command	TX Min	TX Max	Preferred Values	RX Min	RX Max	Preferred Values
255	Factory set-up and test	-	-	-	-	-	-
254	Simple poll	0	0	0	0	0	0
253	Address poll	0	0	0	***var***	***var***	-
252	Address clash	0	0	0	***var***	***var***	-
251	Address change	1	1	1	0	0	0
250	Address random	0	0	0	0	0	0
249	Request polling priority	0	0	0	2	2	2
248	Request status	0	0	0	1	1	1
247	Request variable set	0	0	0	1	255	-
246	Request manufacturer id	0	0	0	3	32	3
245	Request equipment category id	0	0	0	3	16	See Table 1
244	Request product code	0	0	0	3	16	-
243	Request database version	0	0	0	1	1	1
242	Request serial number	0	0	0	3	4	3
241	Request software revision	0	0	0	3	32	-
240	Test solenoids	1	2	1	0	0	0
239	Operate motors	1	1	1	0	0	0
238	Test output lines	1	2	1	0	0	0
237	Read input lines	0	0	0	1	255	-
236	Read opto states	0	0	0	1	255	-
235							
234							
233	Latch output lines	1	2	1	0	0	0
232	Perform self-check	0	0	0	1	2	2
231	Modify inhibit status	1	16	2, 4, 8	0	0	0
230	Request inhibit status	0	0	0	1	16	2, 4, 8
229	Read buffered credit or error codes	0	0	0	11	11	11
228	Modify master inhibit status	1	1	1	0	0	0
227	Request master inhibit status	0	0	0	1	1	1
226	Request insertion counter	0	0	0	3	3	3
225	Request accept counter	0	0	0	3	3	3
224							
223							
222	Modify sorter override status	1	1	1	0	0	0
221	Request sorter override status	0	0	0	1	1	1
220							
219	Enter new PIN number	4	4	4	0	0	0
218	Enter PIN number	4	4	4	0	0	0
217	Request payout high / low status	0	1	0	1	1	1
216	Request data storage availability	0	0	0	5	5	5

215	Read data block	1	1	1	1	255	8
214	Write data block	2	255	9	0	0	0
213	Request option flags	0	0	0	1	1	1
212	Request coin position	1	1	1	1	16	2, 4, 8
211	Power management control	1	1	1	0	0	0
210	Modify sorter paths	2	5	2, 5	0	0	0
209	Request sorter paths	1	1	1	1	4	1, 4
208	Modify payout absolute count	2	3	-	0	0	0
207	Request payout absolute count	0	1	-	2	2	2
206							
205							
204	Meter control	1	4	-	0	3	-
203	Display control	1	255	-	0	2	-
202	Teach mode control	1	2	-	0	0	0
201	Request teach status	1	1	1	2	2	2
200							
199	Configuration to EEPROM	0	0	0	0	0	0
198	Counters to EEPROM	0	0	0	0	0	0
197	Calculate ROM checksum	0	0	0	4	4	4
196	Request creation date	0	0	0	2	2	2
195	Request last modification date	0	0	0	2	2	2
194	Request reject counter	0	0	0	3	3	3
193	Request fraud counter	0	0	0	3	3	3
192	Request build code	0	0	0	3	32	8
191	Keypad control	1	1	1	2	255	-
190							
189	Modify default sorter path	1	1	1	0	0	0
188	Request default sorter path	0	0	0	1	1	1
187	Modify payout capacity	2	3	-	0	0	0
186	Request payout capacity	0	1	-	2	2	2
185	Modify coin id	7	7	7	0	0	0
184	Request coin id	1	1	1	6	6	6
183	Upload window data	2	255	-	0	0	0
182	Download calibration info	0	0	0	1	255	-
181	Modify security setting	2	2	2	0	0	0
180	Request security setting	1	1	1	1	1	1
179	Modify bank select	1	1	1	0	0	0
178	Request bank select	0	0	0	1	1	1
177	Handheld function	1	255	-	0	255	-
176	Request alarm counter	0	0	0	1	1	1
175	Modify payout float	2	3	-	0	0	0
174	Request payout float	0	1	-	2	2	2
173	Request thermistor reading	0	0	0	1	1	1
172	Emergency stop	0	0	0	1	1	1
171	Request hopper coin	0	0	0	6	6	6
170	Request base year	0	0	0	4	4	4
169	Request address mode	0	0	0	1	1	1
168	Request hopper dispense count	0	0	0	3	3	3
167	Dispense hopper coins	4	9	9	0	1	-
166	Request hopper status	0	0	0	4	4	4
165	Modify variable set	1	255	-	0	0	0
164	Enable hopper	1	1	1	0	0	0
163	Test hopper	0	0	0	1	3	3
162	Modify inhibit and override registers	6	6	6	0	0	0
161	Pump RNG	8	8	8	0	0	0
160	Request cipher key	0	0	0	8	8	8
159	Read buffered bill events	0	0	0	11	11	11
158	Modify bill id	8	8	8	0	0	0

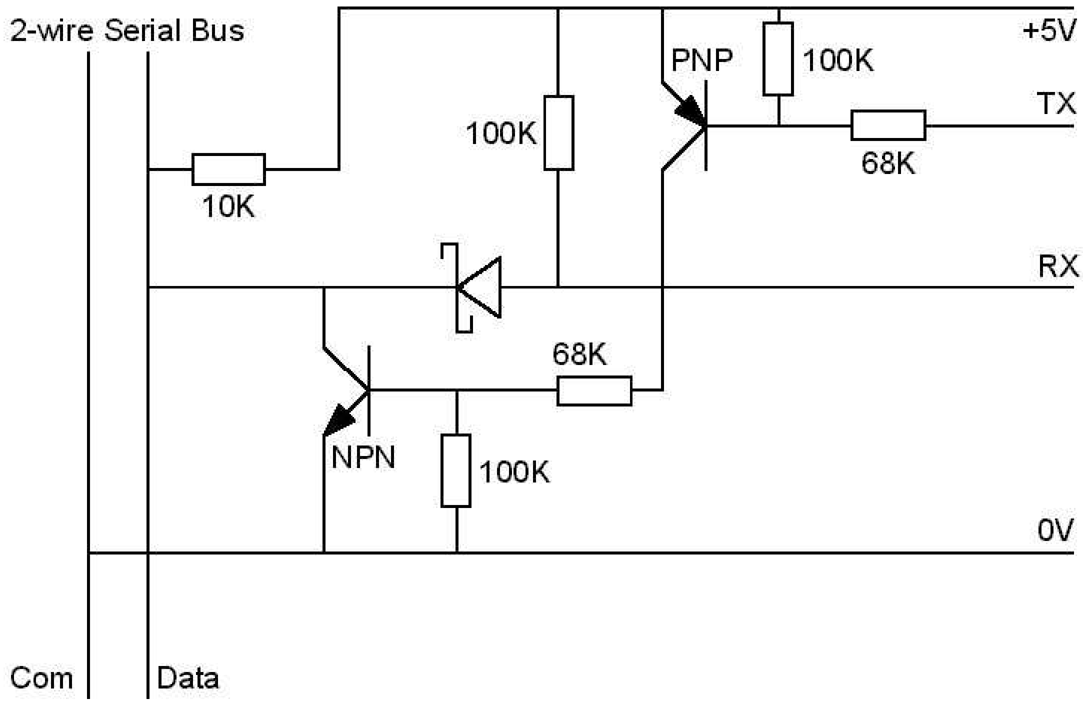
157	Request bill id	1	1	1	7	7	7
156	Request country scaling factor	2	2	2	3	3	3
155	Request bill position	2	2	2	1	16	2, 4, 8
154	Route bill	1	1	1	0	1	-
153	Modify bill operating mode	1	1	1	0	0	0
152	Request bill operating mode	0	0	0	1	1	1
151	Test lamps	2	2	2	0	0	0
150	Request individual accept counter	1	1	1	3	3	3
149	Request individual error counter	1	1	1	3	3	3
148	Read opto voltages	0	0	0	1	255	-
147	Perform stacker cycle	0	0	0	0	1	-
146	Operate bi-directional motors	3	3	3	0	0	0
145	Request currency revision	0	2	-	1	32	3
144	Upload bill tables	3	130	130	0	0	0
143	Begin bill table upgrade	0	0	0	0	0	0
142	Finish bill table upgrade	0	0	0	0	0	0
141	Request firmware upgrade capability	0	1	0	1	1	1
140	Upload firmware	3	130	130	0	0	0
139	Begin firmware upgrade	0	1	0	0	0	0
138	Finish firmware upgrade	0	0	0	0	0	0
137	Switch encryption code	3	3	3	0	0	0
136	Store encryption code	0	0	0	0	0	0
135	Set accept limit	1	1	1	0	0	0
134	Dispense hopper value	10	10	10	0	1	-
133	Request hopper polling value	0	0	0	7	7	7
132	Emergency stop value	0	0	0	2	2	2
131	Request hopper coin value	1	1	1	8	8	8
130	Request indexed hopper dispense count	1	1	1	3	3	3
129	Read barcode data	0	0	0	0	32	-
128	Request money in	0	0	0	4	4	4
127	Request money out	0	0	0	4	4	4
126	Clear money counters	0	0	0	0	0	0
125	Pay money out	4	4	4	0	0	0
124	Verify money out	0	0	0	9	9	9
123	Request activity register	0	0	0	2	2	2
122	Request error status	0	0	0	2	2	2
121	Purge hopper	2	2	2	0	0	0
120	Modify hopper balance	3	3	3	0	0	0
119	Request hopper balance	1	1	1	8	8	8
118	Modify cashbox value	4	4	4	0	0	0
117	Request cashbox value	0	0	0	4	4	4
116	Modify real time clock	4	4	4	0	0	0
115	Request real time clock	0	0	0	4	4	4
114	Request USB id	0	0	0	4	4	4
113	Switch baud rate	2	2	2	0	1	-
112	Read encrypted events	1	1	1	16	16	16
111	Request encryption support	6	6	6	17	17	17
110	Switch encryption key	16	16	16	0	0	0
109	Request encrypted hopper status	3	3	3	16	16	16
108	Request encrypted monetary id	2	2	2	16	16	16
107	Operate escrow	1	1	1	0	0	0
106	Request escrow status	0	0	0	3	3	3
105	Data stream	4	5	-	0	255	-
105	Request service status	1	1	1	0	1	-
004	Request comms revision	0	0	0	3	3	3
003	Clear comms status variables	0	0	0	0	0	0

002	Request comms status variables	0	0	0	3	3	3
001	Reset device	0	0	0	0	0	0

20 Circuit 1 - ccTalk Standard Interface

Note that the original design using a PNP receive transistor has been abandoned due to the data line voltage on some products falling to nearer +4V than +5V. The PNP design did not give enough safety margin.

This circuit uses an open-collector transistor to drive the data line and a diode protected straight-through receiver.

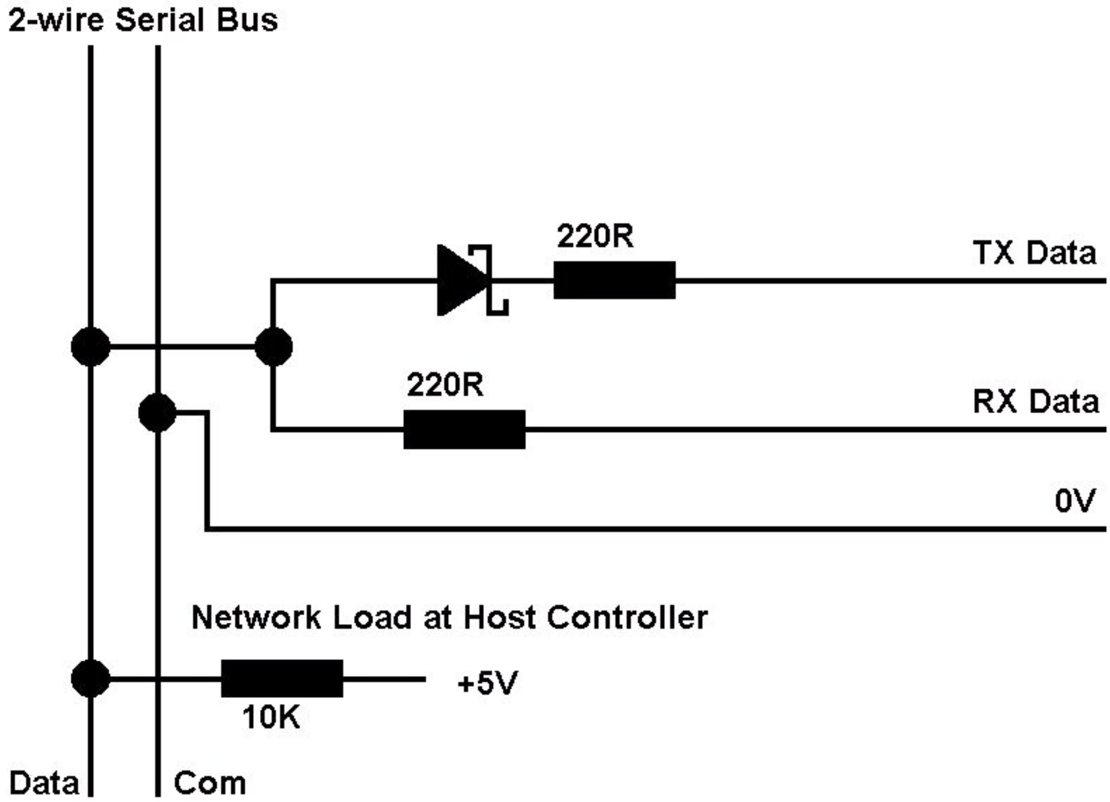


20.1 Typical Components

Diode	BAT54	Schottky Diode, low forward voltage drop
NPN	BC846B	High gain, medium signal, NPN transistor
PNP	BCW68	High gain, medium signal, PNP transistor

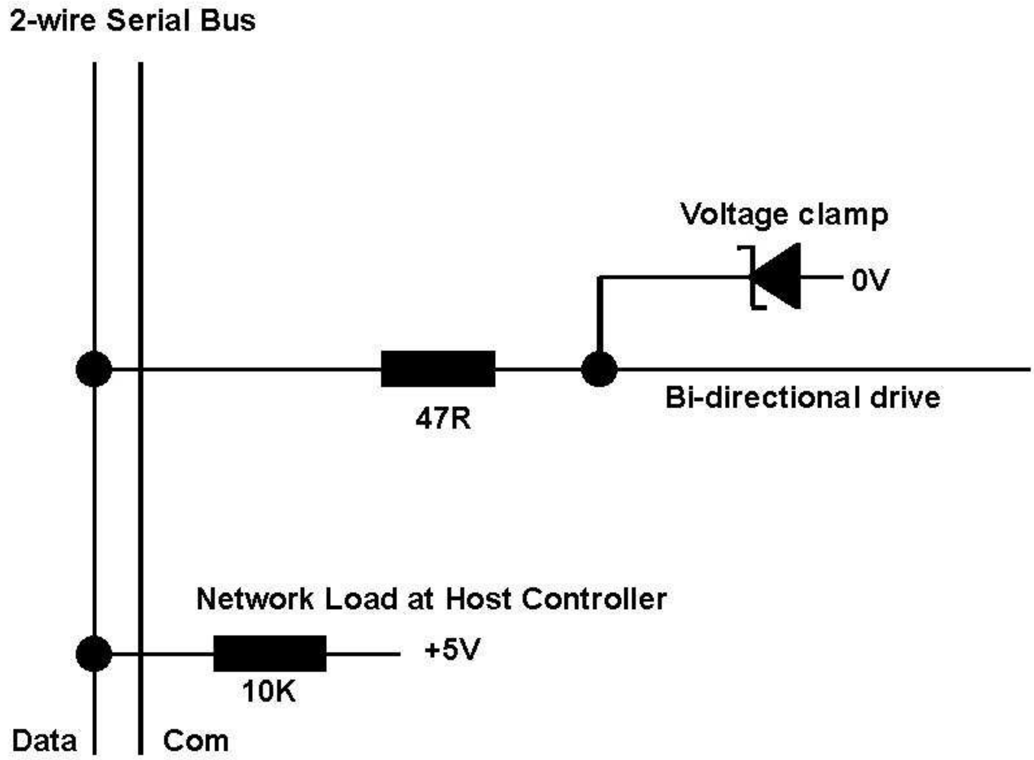
21 Circuit 2 - ccTalk Low Cost Interface

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



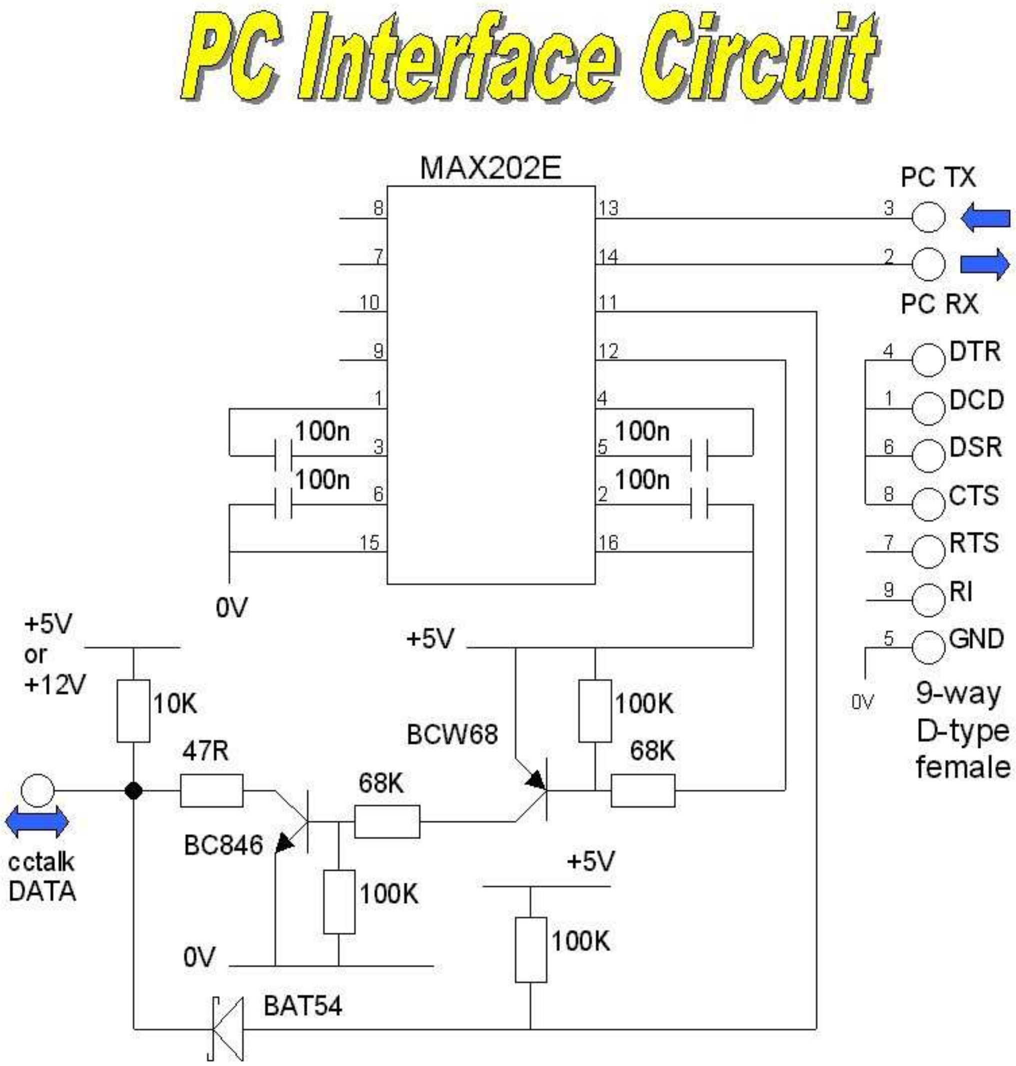
22 Circuit 3 - ccTalk Direct Interface

A very low cost solution is to interface a single pin on a microcontroller directly onto the ccTalk data line. The pin can be switched between active-low for transmitting and high-impedance tri-state for receiving.



23 Circuit 4 - PC Interface

The circuit below shows how to connect the 9-pin serial port of a PC to the ccTalk data bus. The only integrated circuit required is a Maxim level-shifter which operates off a single +5V supply. Any small-signal diodes and transistors can be used.



If you want non-smt then…
A direct equivalent to BCW68 (PNP) is BC327 in a TO-92 package.
A direct equivalent to BC846 (NPN) is BC546 in a TO-92 package.

Transistors are any small signal, general-purpose types with a dc current gain of at least 100.

24 Glossary

Presented below is an arbitrary selection of terms relating to serial communications and the money transaction industry which may prove helpful to people unfamiliar with this field.

Accumulator	As in ‘accumulator hopper’. The hopper can dispense multiple coin types to reach a requested coin value but can only determine which coin has been paid after it leaves the hopper. To prevent overpaying the hopper must sometimes stop before the requested coin value is met and inform the host machine so that a second single-coin hopper can dispense the remaining coin value or ‘change’. This method can be substantially faster than a discriminator hopper which rejects unwanted coin types before they leave the hopper.
ACK	Acknowledge message. Affirmative outcome, it worked.
API	Application Program Interface. The use of a common software library through standardised ‘hooks’.
ASCII	American Standard Code for Information Interchange. A universal way of representing letters with numbers
Asynchronous	Data is transferred at seemingly random intervals, i.e. not synchronised with a master clock.
AWP	Amusement With Prize - a type of amusement machine. Typically reels.
BACTA	British Amusement Catering Trade Association (founded 1974). Represents the pay-to-play leisure industry in Great Britain.
Bit	(Binary Digit). The smallest unit of digital information - a 0 or 1.
Bit stuffing	The process of adding extra bits into a transmitted packet to ensure continuous data transfer
BNV	Bank Note Validator
Broadcast	Sending a message to all bus peripherals regardless of address
Bus	The electrical connection along which data flows between devices
Byte	(Binary Term). A storage location for 8 bits
Calibration	The Money Controls method of remote coin programming
CAN	Controller Area Network - an automotive serial protocol
CCITT	Comité Consultatif International Téléphonique et Télégraphique
Checksum	A method of detecting errors in transmitted data
CRC	Cyclic Redundancy Check - a secure type of checksum based on polynomial division
CSMA/CD	Carrier Sense Multiple Access / Collision Detection. A method of handling collisions on a multi-master network.
CVF	Coin Value Format as used by ccTalk
Discriminator	As in ‘discriminator hopper’. The hopper can dispense multiple coin types. The hopper determines each coin type prior to payout and if it does not match the required type it is ‘rejected’. Can suffer from ‘hunting’ whereby the hopper cannot find a particular coin if the frequency of occurrence is low.
EEPROM	Electrically Erasable Read Only Memory. Non-volatile storage.
EMS	Early Morning Start-up. Software executed when power is first applied to a machine. Traditionally assumed to be first thing in the morning.
EIA	Electronic Industries Association
Ethernet	A common networking protocol employing CSMA/CD on a bus or star topology
Full-duplex	Data can be transmitted and received simultaneously
Half-duplex	Data can be transmitted and received, but not simultaneously
IEEE	Institution of Electrical and Electronic Engineers
ISO	International Standards Organisation
Isochronous	Data is transferred subject to some time constraints. Applications such as multi-media must have a guaranteed data throughput.
ITU	International Telecommunication Union
MDCES	Multi-Drop Command Extension Set as used by ccTalk
Mech	Industry-standard abbreviation for a coin (mech)anism
MPU	Micro-processing Unit. Old-fashioned word for a PCB with a microprocessor on it. A system block with processing capability.

Multi-Drop	More than one slave device on a common bus
NAK	No Acknowledge message. Negative outcome, it failed.
Nibble	Alternatively nybble. 4 bits of information or half a byte.
Non-volatile	Data retained after power is removed
NRZ	Non return to zero. Some protocols allow continuous streams of 0's and 1's to be sent out - the NRZ method. Other protocols require forced bit transitions to recover the clock signal.
Open-collector	A method of driving a signal between ground and high impedance. Ideal for multi-drop networks.
OSI	Open System Interconnection. An ISO standard for networking.
Parity	A method of testing for a single bit error in a data packet by counting the number of set bits. Can be an odd or even parity check depending on the calculation.
Protocol	A set of common rules to allow devices to communicate
RAM	Random Access Memory. Read / write memory for data storage.
RNG	Random Number Generator
ROM	Read Only Memory. Fixed memory, usually for the program code itself.
RS232C	Recommended Standard 232C - the original serial standard for data communications
RS485	Similar to RS232 but multi-drop and long distance
RTBY	Relative To Base Year. A Money Controls date format.
Start bit	Used to signal the start of a data packet and initiate any timing control. Essential in asynchronous communications
Stop bit	Used to signal the end of a data packet
String	A sequence of printable characters
SWP	Skill With Prize - a type of amusement machine. Typically a quiz.
Synchronous	Data is transferred at regular intervals in time according to a master clock
Topology	The shape of a network - how the devices are physically distributed and connected together.
UART	Universal Asynchronous Receiver Transmitter. The portion of hardware which transfers data to and from a bit stream.
USB	Universal Serial Bus - a PC peripheral protocol
Validation	The process of recognising a coin or bill as the genuine article