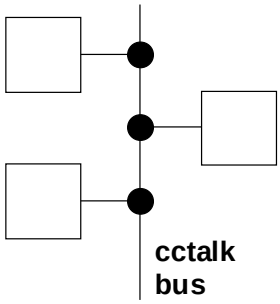


cctalk Expansion for Bill Validators

Issue 2.4

Revision History

Issue	Date	Comments
1	15/03/00	First draft
1a	17/03/00	Extra text clarification Notation section added Lamp no. added to 'Control front-panel lamps' Speed option added to 'Operate bi-directional motors'
1b	22/06/00	Change to connector type New command header numbers have been **CHANGED** Some text modifications
2	09/08/00	In response to BACTA sub committee meeting… Addition of CRC checksum Addition of security layer Change to connector type
2.1	02/11/00	Typo corrected on page 29. Header 150 = Request individual accept counter Header 149 = Request individual error counter (command list summary was correct)
2.2	27/11/00	Addition of commands… Header 239 = Operate motors Header 236 = Read opto states Header 195 = Request last modification date "Unknown" return string added to Request currency revision Upload bill tables - clarification on bill type New chapter : Switching and Storing Encryption Codes Addition of glossary
2.3	23/04/03	Added the TSP number Changed headers and footers Re-paginated
2.4	30/06/04	Changed footer

Contents

1. Introduction

This document covers the expansion in command set needed to operate a bill validator on the cctalk serial interface.

This document is not meant to be read alone. More details of the cctalk protocol are given in the generic specification which includes information on the packet structure and error handling. However, the protocol has now been expanded to cover CRC checksum and encryption layers.

2. Protocol Settings

2.1. General

Various options exist on cctalk products but the following will be adopted on all bill validators to maintain compatibility between manufacturers.

Bill Validators
cctalk b96.p0.v12.a12.d0.c5.m0.x16.e1.i0.r4

9600 baud
Open-collector interface
Nominal supply voltage +12V
Serial data pull-up voltage +12V
Supply sink
Connector type 5 (10-way dual header)
Slave device only
CRC CCITT checksum
Encryption type 1
Minor release 0
Major release 4

2.2. Address

The default address for bill validators is **40**.

2.3. Electrical Interface

The cctalk data line requires an open-collector drive. A pull-up resistor to +Vs of 10K should be provided on the bill validator.

3. Connector Pinout

(1) /DATA ← cctalk interface
(2) -
(3) -
(4) -
(5) /RESET optional use in cctalk
(6) -
(7) +Vs ← cctalk interface
(8) 0V ← cctalk interface
(9) /SERIAL MODE ← cctalk interface, connect to 0V for serial operation
(10) -

Pin 9 may be used to switch the unit into cctalk serial mode when alternative protocols (especially parallel) are supported.

4. Connector Type

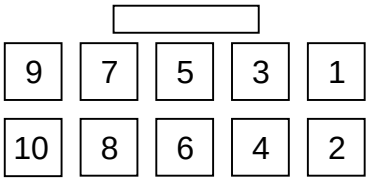
Recommended peripheral connector :
Molex 8624 Series 0.1inch dual row straight pin breakaway header
P/N 10-89-1101 (15µ gold)

Mechanical keying should be provided by the surrounding cover.

(Alternative :
Molex 70246 Series dual row straight pin low profile shrouded header 70246-1021)

Mates with :
Molex 40312 Series MX50 ribbon cable connector system
P/N 15-29-9710 (15µ gold, centre polarisation, strain relief)

Pin Polarity :



View of connector from front

5. cctalk Bill Features

cctalk offers a secure, flexible and easy to implement protocol for connecting bill validators to host machines. Multi-drop capability means that coin acceptors and hoppers can be connected to the same serial bus.

The most important features of this protocol for bill validators are…

- Simple, high-level, command structure operating at 9600 baud. No address bits or parity bits required. Genuine 8-bit data packets throughout.
- CRC checksums for data integrity. Even the 'ACK' message is protected.
- Encryption on all commands
- Secure operation. A fault with the host machine polling or the serial data line will automatically prevent bill acceptance
- Support for stacker operation
- Support for escrow operation
- Base support for 16 different bill types. Can be easily expanded to 256 banks of 64 bill types = 16,384 bill types. Bills can be split in any arbitrary way between currencies e.g. Germany and Euro.
- Bill country and value can be downloaded into a host 'RAM table' during initialisation.
- Bill values can range from 0.01 to over half a billion - no practical restriction.
- Support for remote bill programming
- Support for flash upgrading of firmware
- Support for bill security tuning
- Support for bill teaching
- Extensive diagnostic support
- Full stacker status reporting e.g. stacker removed, stacker inserted…

6. Bill Validator Extensions

The bill validator will conform to cctalk generic specification issue 3.2 (as per 3.1 with serial hopper commands) but will have extensions to cover CRC checksums and encryption. Once agreed by all parties, these extensions will be incorporated into generic specification issue 4.0.

CRC's have been added to allow detection of all double-bit errors which is appropriate given the high denominations of bills which will be transferred across the bus.

An encryption layer has been added into the protocol to guard against an illegal 'listening device' connected to the data bus being used to de-fraud the host machine. Each bill validator will require its own security key to operate.

The extensions to the cctalk protocol have been made without changing the overall packet length and maintaining the 'Destination Address / No. of Data Bytes' start tag. This maintains a degree of compatibility with the low-level cctalk drivers which already exist, and allows coin acceptors, serial hoppers and bill validators to co-exist on the same bus.

6.1. Compatibility with other cctalk Peripherals

Bill validators using CRC checksums and encryption may be mixed with other cctalk peripherals which don't have these additional features due to the compatible packet structure. However, it will be necessary to avoid the use of the broadcast message (destination address = 0) as one set of peripherals will inevitably decode the command headers and checksums incorrectly. At best, the peripheral will register a checksum fail. At worst, it will execute the wrong command. The only useful broadcast address commands are the MDCES commands 'Address poll' and 'Address random' which can be used to scan the bus addresses and randomise them all in the event of a clash. Therefore, these commands have to be avoided and the peripheral addresses known in advance by the host machine. This will not be a problem as we are defining the bill validator as address 40 and the coin acceptor and serial hoppers will likewise be at known addresses.

7. CRC Checksum

7.1. CRC Packet Structure

All cctalk bill validators should support CRC checksums. The standard cctalk message packet is modified as follows…

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

We lose the 'Source Address' field and replace it with the lower byte of the 16-bit CRC code. **The source address of the host machine will always be assumed to be '1'**.

The checksum is calculated for all message bytes i.e. destination address + no. of data bytes + header + any data bytes.

Various confusing options exist for CRC algorithms. The one cctalk will adopt is the most common…

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

See Appendix A for the algorithm and test data.

7.2. Exceptions

The responses from the MDCES commands 'Address poll' and 'Address clash' are single bytes with no checksum data.

8. Encryption

8.1. Encryption Mechanism

There is an option (always enabled on bill validators) to encrypt the data in a cctalk message packet. No additional data bytes are sent to maintain bus performance - just a scrambling of the existing ones.

The [Destination Address] and [No. of Data Bytes] bytes are left unchanged as they are needed by the bottom layer of the cctalk driver software. The subsequent bytes in both transmit and receive messages are encrypted. This includes the header and checksum bytes.

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

8.2. Encryption Algorithm

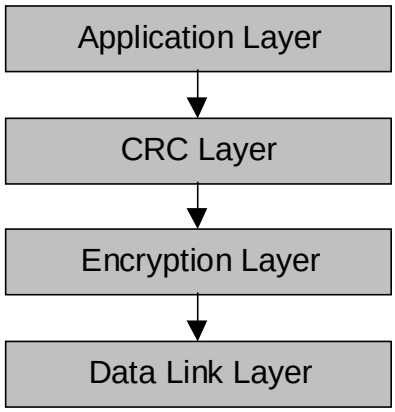
The encryption algorithm will be made available in another document and will be subject to controlled release. The algorithm will use a 6-digit ‘security code’ which will have to be entered into the host machine prior to using a new unit.

8.3. Exceptions

The responses from the MDCES commands ‘Address poll’ and ‘Address clash’ are single bytes with no encryption.

9. Protocol Layering

9.1. General Format



The encryption layer is placed below the CRC layer to give extra security. Otherwise a message could be intercepted, the data changed and the checksum re-calculated. The decrypted command would have a 'random' header and be a potential security risk.

9.2. TX Messages

The procedure is outlined as follows…

- Prepare data packet from destination address, header & data bytes
- Calculate CRC checksum and insert into structure
- Apply encryption to [N - 2] message bytes

9.3. RX Messages

The procedure is outlined as follows…

- Receive message bytes. If address valid then 'receive and store' else 'receive and count'
- Decrypt message
- Calculate CRC checksum and check to see if data is valid
- Action according to header

10. Security Features and Typical Operation

At power-up or reset, all bills are automatically inhibited. Polling the bill validator will return an event counter of zero. The host machine can then download the different bill types available and create a look-up table for performing vends or game credits. To begin operation, the individual bill inhibits (sometimes referred to as 'lockouts') are released according to the currencies which wish to be accepted. Then finally, the master inhibit is released to allow customers to insert bills.

Bill credit information is polled out of the unit typically every 200ms. Since credit information is sent as a history buffer, any checksum or other comms error allows re-transmission to take place safely without losing any credit information.

The host **must maintain a minimum polling rate** to ensure correct operation of the money handling systems. If the host does not check for bill acceptance very often in escrow mode, the customer can be kept waiting with a bill in the machine.

If the host fails to poll the bill validator for whatever reason, the 'credit poll timeout' expires and the unit is placed in master inhibit mode. A 'Master inhibit active' event is written to the history buffer to inform the host when it next resumes polling. The host will then have to release the master inhibit condition before normal operation resumes. This timeout period may typically be 2s.

If the host machine cannot maintain the minimum polling rate due to other system requirements, it can put the bill validator to 'sleep' with the master inhibit command. This prevents the bill validator from accepting further bills until the host is ready. The credit poll timeout will be suspended during this interval. If a bill is already in the validator when the master inhibit command is issued then a credit event is still generated and will be available when polling resumes. Since after the bill validator is put in master inhibit mode there may be a pending credit so it is not recommended that the host suspends polling for any great length of time. **Frequent polling is a requirement of the system.**

If escrow mode is not being used then a bill credit is generated only after the bill is sent to the cashbox or stacker. If escrow mode is being used then a bill escrow event is generated, after which the host machine can decide whether to return the bill or send it to the cashbox / stacker. If no instruction is received by the bill validator within an 'escrow timeout' period (typically 2 minutes) then the bill is returned to the customer.

If the stacker is full or faulty then no bills can be accepted by the validator. After the last bill is stacked, a 'Stacker full' event is produced. If the stacker is then removed whilst on-line, a 'Stacker removed' event is produced. Plugging it back in gives a 'Stacker inserted' event. If the stacker is no longer full, then a 'Stacker OK' event is generated. The number of bills in the stacker at any given time is not reported by the bill validator but relative counts can be performed by the host machine.

11. Identification Strings

Due to a number of cctalk bill validators appearing on the market with different feature sets, some method of identification is essential. The following cctalk commands can be used.

Command	Example	Comments
Request equipment category id	"Bill Validator"	Must be this !
Request product code	"Lumina"	Manufacturer's product code
Request build code	"Standard"	Any product options
Request manufacturer id	"MCI"	The manufacturer's identification
Request serial number	12,345,678	A 24-bit binary number
Request software revision	"LM-V1.00"	The software / firmware revision
Request comms revision	[1] [4] [0]	First release of cctalk spec. 4.0
Request currency revision	"GB-V1.00"	The currency revision

12. Host Software - Simplified Outline

The host software can be as complicated as a full cctalk implementation allows. There is a big difference in writing software for a single bill validator of known type compared with a full-blown multi-drop bus application with a number of peripherals made by different manufacturers.

A simplified algorithm for host software is shown below, without showing the extra code needed to detect unexpected power removal, poll timeouts, jams etc.

The code related to security and encryption is not shown here.

Power-up cctalk bus

Confirm and identify standard bus addresses with…
- 'Simple poll'
- 'Request equipment category id'

Confirm support available for a particular model…
- 'Request product code'
- 'Request build code'
- 'Request manufacturer id'
- 'Request comms revision'

'Perform self-check' - are there any faults ?
'Request option flags' - check feature set
'Modify bill operating mode' - select stacker / escrow options

For each bill type
- 'Request bill id' - generate host look-up table of bill types

For each country type
    'Request country scaling factor' - store scaling factors

'Modify inhibit status' - select bills of interest
'Modify master inhibit status' - begin bill acceptance

Do
    'Read buffered bill events'

    If 'bill in escrow'
        'Route bill'
    else if 'bill sent to stacker'
        Vend or Credit
    else if 'bill returned'
        do nothing
While True

The last loop is the main software polling loop of the program. If a coin acceptor is also connected to the bus then that will be polled in the same loop as well.

13. Manufacturer-Specific Commands

Any manufacturer-specific commands not covered in this document will be provided in the usual way through command header 255. Commands accessed through this header number are typically not available to customers and may have some kind of password protection.

There is no practical limit to the number or data format of manufacturer-specific commands.

13.1. Bill Identification

At its simplest level, a bill credit will be identified by an integer value, e.g. 1 to 16. So looking back at a credit log, we may have…

Credit 2
Credit 5
Credit 4
Credit 4
Credit 3
Credit 1
…

A standard may be published by BACTA which covers the credit codes and lockout positions for UK AWP.

For example…

Inhibit / Lockout Position	Credit Code	Bill Type
1	1	GB £20.00
2	2	GB £10.00
3	3	GB £5.00

More sophisticated implementations could use the following commands to identify the note type regardless of its position within the bill validator.

Header 157 - Request bill id
Header 156 - Request country scaling factor

In this case, there would be no need to standardise on lockouts etc.

More information on the use of these commands is detailed below.

14. Command List Summary

Most commands below are already described in the generic specification and are directly applicable to bill validators. The command set is 'feature rich' and certain manufacturers may choose not to implement them in their entirety. The recommended policy for replying to unimplemented command headers is don't do it - i.e. do not return any message bytes at all. If there is no reply, the command attempted clearly does not work.

It is recommended that all manufacturers of cctalk bill validators support the commands below marked with an asterisk '*'

14.1. Core Commands

***254 - Simple poll**
***245 - Request equipment category id**
***244 - Request product code**
***192 - Request build code**
***246 - Request manufacturer id**

14.2. Core Plus Commands

***242 - Request serial number**
***241 - Request software revision**
***004 - Request comms revision**
002 - Request comms status variables
003 - Clear comms status variables
***001 - Reset device**
197 - Calculate ROM checksum
169 - Request address mode

14.3. Multi-drop Commands

253 - Address poll
252 - Address clash
251 - Address change
250 - Address random

14.4. Bill Validator Commands

***159 - Read buffered bill events**
***231 - Modify inhibit status**
***230 - Request inhibit status**
***228 - Modify master inhibit status**
***227 - Request master inhibit status**
179 - Modify bank select
178 - Request bank select
181 - Modify security setting
180 - Request security setting
158 - Modify bill id
157 - Request bill id
156 - Request country scaling factor
155 - Request bill position
***154 - Route bill**
***213 - Request option flags**
***153 - Modify bill operating mode**
***152 - Request bill operating mode**
226 - Request insertion counter
225 - Request accept counter
150 - Request individual accept counter
149 - Request individual error counter
***247 - Request variable set**
249 - Request polling priority
202 - Teach mode control
201 - Request teach status
196 - Request creation date
195 - Request last modification date
170 - Request base year
***216 - Request data storage availability**
215 - Read data block
214 - Write data block
***145 - Request currency revision**
144 - Upload bill tables
143 - Begin bill table upgrade
142 - Finish bill table upgrade
***141 - Request firmware upgrade capability**
140 - Upload firmware
139 - Begin firmware upgrade
138 - Finish firmware upgrade
***137 - Switch encryption code**
***136 - Store encryption code**

14.5. Diagnostic Commands

```
*232 - Perform self-check
 237 - Read input lines
 238 - Test output lines
 233 - Latch output lines
 151 - Test lamps
 236 - Read opto states
 148 - Read opto voltages
*147 - Perform stacker cycle
 239 - Operate motors
 146 - Operate bi-directional motors
```

15. Notation

The ‘Transmitted data’ and ‘Received data’ fields only show data bytes - the rest of the cctalk packet structure is not shown for ease of reading.

Transmitted data bytes, indicated by square brackets, are shown in the order they are transmitted, i.e. the first byte transmitted on the left, the last byte transmitted on the right.

Received data bytes, indicated by square brackets, are shown in the order they are received, i.e. the first byte received on the left, the last byte received on the right.

Data byte values are shown in decimal unless otherwise stated.

Bit fields within bytes are shown the other way around - with the MSB on the left and LSB on the right.

For example…
Received data : [B1:7 … B1:0] [B2:7 … B2:0] [B3:7 … B3:0]

B1:7 = bit 7 of byte 1, the first byte received
B3:0 = bit 0 of byte 3, the last byte received

If this data was a counter value, then it could be converted into decimal by calculating…
B1 + (256 * B2) + (65536 * B3)

Any fields which show [bill type] refer to the ‘number’ or ‘channel’ of the bill. Different manufacturers use different terms. For instance, if a bill validator recognises 16 different bills, then the bill type is a number from 1 to 16. Type 1 may be a $1 bill, type 2 may be a $10 bill etc… The bill type does not reflect orientation information - each type may have up to 4 possible orientations, the profiles of which are stored internally.

The bill type also determines which inhibit bit is used to inhibit or enable the bill with the ‘Modify inhibit status’ command. Using a 16-bit inhibit mask, inhibit 1 controls bill type 1, inhibit 2 controls bill type 2… inhibit 16 controls bill type 16. There is no option to re-map these inhibits into a different pattern.

16. Core Commands in Detail

16.1. Header 254 - Simple poll

Transmitted data : <none>
Received data : ACK

16.2. Header 245 - Request equipment category id

Transmitted data : <none>
Received data : “Bill Validator”

Any manufacturer’s equipment previously described as a ‘Bill Acceptor’ or ‘Note Acceptor’ will adopt the MDB naming convention of ‘Bill Validator’. Failure to report the exact characters ‘Bill Validator’ will mean the host machine will not attempt to operate the device.

Note that in line with other cctalk commands, the string is case-sensitive.

In decimal, the return data is…

```
[ 66 ] [ 105 ] [ 108 ] [ 108 ] [ 032 ]
[ 86 ] [ 097 ] [ 108 ] [ 105 ] [ 100 ] [ 97 ] [ 116 ] [ 111 ] [ 114 ]
```

16.3. Header 244 - Request product code

Transmitted data : <none>
Received data : ASCII string

A manufacturer-specific product code e.g. “Lumina”

16.4. Header 192 - Request build code

Transmitted data : <none>
Received data : ASCII string

The build code is an ASCII string (typically 8 characters or less) representing important product build features. This string is manufacturer-specific so consult product documentation for more details.

If a manufacturer chooses not to implement this feature then they should return the string ‘Standard’ as a generic device response.

16.5. Header 246 - Request manufacturer id

Transmitted data : <none>
Received data : ASCII string

See Appendix E for a list of registered bill validator manufacturers.

If you are not on this list and would like to be then please contact Money Controls.

The bill validator will return a **3 character ASCII string** so that the host machine can identify any manufacturer-specific command features.

17. Core Plus Commands in Detail

17.1. Header 242 - Request serial number

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

All bill validators are programmed sequentially with a 3 byte serial number during the factory set-up process. This serial number cannot be subsequently changed.

The serial number is stored in 'machine readable' or binary format, rather than ASCII. The least significant byte is returned first.

For example, if the return data is [78] [97] [188] then the serial number is
78 + 256 * 97 + 65536 * 188 = 12,345,678.

17.2. Header 241 - Request software revision

Transmitted data : <none>
Received data : ASCII string

e.g. "LM-V1.00"

Any internal changes in firmware will be represented in the software revision string. It is not recommended that any code changes, however small, are 'hidden' from external examination.

17.3. Header 4 - Request comms revision

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

All bill validators will return at least [1] [4] [0] to reflect the new issue 4.0 specification document.

A manufacturer may wish to up-issue the 'cctalk level' to reflect any comms driver changes that need to be notified to the host machine.

17.4. Header 2 - Request comms status variables

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

Operation as per the generic specification.

17.5. Header 3 - Clear comms status variables

Transmitted data : <none>
Received data : ACK

Operation as per the generic specification.

17.6. Header 1 - Reset device

Transmitted data : <none>
Received data : ACK

Operation as per the generic specification.

The product documentation should state clearly the time required for the bill validator to initialise and be ready to accept the next serial command.

17.7. Header 197 - Calculate ROM checksum

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

The unit will calculate and return a 4 byte checksum of the firmware in the bill validator. The method of calculation and address ranges used are manufacturer-specific.

[checksum 1] is the least significant byte.

This command does not confirm whether the checksum is 'correct' but BNV fault code 30 can be used for this purpose.

17.8. Header 169 - Request address mode

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

Refer to the generic specification.

A typical product may return 84H to indicate that the device address is stored in EEPROM and may be changed using serial commands.

18. Multi-drop Commands in Detail

18.1. Header 253 - Address poll

Operation as per the generic specification.

18.2. Header 252 - Address clash

Operation as per the generic specification.

18.3. Header 251 - Address change

Operation as per the generic specification.

18.4. Header 250 - Address random

Operation as per the generic specification.

19. Bill Validator Commands in Detail

19.1. Header 159 - Read buffered bill events

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

[event counter]
0 = power-up or reset
1 to 255 = event counter

After 255, the event counter loops back to 1.

A history buffer of the last 5 events is always returned. Each event is a 2 byte pair, representing a bill credit or an error / event code.

Credits

[1 to 255] [0]	Bill type 1 to 255 validated correctly, sent to cashbox / stacker
[1 to 255] [1]	Bill type 1 to 255 validated correctly, held in escrow

If an escrow is fitted and enabled, then bills are kept in the escrow until a host command is issued ('Route bill') instructing the unit to return the bill or to send the bill to the cashbox or stacker. If this is the case, a host vend or credit should not take place until confirmation is received of this manual accept instruction.

Errors / Events

[0] [0]	Master inhibit active
[0] [1]	Bill returned from escrow

See Appendix C (BNV Event Codes) for the rest.

When the bill validator powers up, or is reset, the event counter is zero, the master inhibit is active and all individual bills are inhibited.

If at any time a credit poll timeout occurs (no 'Read buffered bill events' have been sent for several seconds) then the bill validator sets the master inhibit and refuses to take any more bills. This is a safety feature in case the host link goes down.

Although individual bills can be inhibited and enabled on serial, there may be some master override DIP switches on the casing. These can be used to inhibit a problem bill without having to change the host machine firmware. If a bill is inhibited by DIP switch then this is reported as such, rather than the usual 'Inhibited bill (on serial)' message. A DIP switch inhibit **cannot** be re-enabled over the serial line.

The 'Bill returned from escrow' message is sent if an escrow is being used and the host machine instructs the bill validator to return rather than accept the bill. If the host fails to instruct the bill validator within a timeout period then the bill is automatically returned to the customer and this message issued.

The normal event for an unrecognised bill type is the 'Invalid bill (due to validation fail)' message. The bill is automatically returned to the customer.

Note that continuous fraud or jam conditions need only be reported once. If they are eventually cleared then the bill validator will report credits as normal.

19.2. Header 231 - Modify inhibit status

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

Bit format : 0 = inhibit bill, 1 = enable bill

There is individual inhibit support for 16 different bill types (2 byte inhibit mask).

All bills default on power-up or reset to 'inhibited'.

Inhibit status is RAM-based and volatile.

Note : Future products may see mask expansion to cover 64 or 128 note types.

19.3. Header 230 - Request inhibit status

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

Note that this command reads the status of the 'serial' inhibits, i.e. those set with the 'Modify inhibit status' command. Any inhibit overrides set on DIP switches are not visible with this command, even if they result in a bill being inhibited.

To read the status of DIP switches, it is suggested that the 'Read input lines' command is used which will obviously be product specific. Some bill validators may have no DIP switches and others may provide a manual override on each and every bill.

19.4. Header 228 - Modify master inhibit status

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

Bit format : 0 = inhibit all bills, 1 = resume normal operation

Only 'Bit 0' is used.

This command allows the bill validator to be quickly placed in a state whereby it inhibits all bills without changing the status of the individual bill inhibits. Therefore, normal operation of the bill validator can be suspended or resumed with this command.

All bill validators default to master inhibit active on power-up or reset.

Master inhibit status is RAM-based and volatile.

To inhibit a bill validator, send [0]
To enable a bill validator, send [1]

19.5. Header 227 - Request master inhibit status

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

See above.

19.6. Header 179 - Modify bank select

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

To allow the manufacture of a bill acceptor with substantially more bill types than 16, it may be beneficial to split the stored types into ‘banks’ and use this command to switch them in and out of memory. Any bills in memory and ‘active’ can be accepted, subject to the status of the individual bill inhibits.

If a bill validator can only support one active bank of 16 bills then the only valid bank number is ‘0’. In which case, this command is redundant and need not be used.

The bank select code is RAM-based and volatile.

19.7. Header 178 - Request bank select

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

See above.

19.8. Header 181 - Modify security setting

Transmitted data : [bill type] [security setting]
Received data : ACK

The bill validator may have a mechanism to change the security setting of various bills to combat a particular fraud problem. Refer to the generic specification for more details - operation for coin acceptors is described. The command is the equivalent of a digital tuning potentiometer giving precise and fine adjustment of close frauds.

The bill security setting is usually RAM-based such that factory defaults are restored after the next power-down. There may be an option to prevents security tuning in sensitive applications.

19.9. Header 180 - Request security setting

Transmitted data : [bill type]
Received data : [security setting]

See above.

19.10. Header 158 - Modify bill id

Transmitted data : [bill type] [char 1] [char 2] [char 3]… [char 7]
Received data : ACK

This command allows the identifier string for a particular bill to be changed - useful after teaching it a new bill.

See below for details.

Bill id strings are stored in EEPROM or NVRAM and are permanent.

19.11. Header 157 - Request bill id

Transmitted data : [bill type]
Received data : [char 1] [char 2] [char 3]… [char 7] e.g. “US0100A”

Each bill type is identified by a 7 character ASCII string. This is formed as follows…

2 character international country code	e.g. US
4 character value code	e.g. 0100
1 character issue code	e.g. A, B, C…

All countries of the world are given a 2 character country code. See Appendix F.

Old and new notes in a currency would have the same country and value codes, but a different issue code. Identifying them uniquely allows an old note to be inhibited at some point and only the new note accepted.

The actual monetary value is in multiples of the scaling factor appropriate to that country - see the next command.

Note that each bill is usually stored internally as 4 different profiles representing the 4 possible insertion orientations. This extra information is not reported externally by ‘customer’ commands. This version of the specification does not allow control over bill orientation. For instance, standard inhibit control is provided for 16 different bills rather than the 64 possible bill-orientation combinations.

19.11.1. Unprogrammed bills

If a bill position is blank, i.e. a bill has not been programmed into that position, then the convention is to return ASCII ‘dots’ instead. Therefore, return ‘…….’ or 7 characters of ASCII code 46.

19.11.2. RAM Table

It is suggested the host machine fill a RAM table of bill types during the bill validator initialisation routine. The ‘Request bill id’ command can be used in conjunction with the ‘Request country scaling factor’ command to determine the exact monetary value of each note.

For example…

Bill type	Bill id	Scaling factor	Decimal places	Value
1	US0001A	100	2	$1.00
2	US0002A	100	2	$2.00
3	US0005A	100	2	$5.00
4	US0010A	100	2	$10.00
5	US0020A	100	2	$20.00
6	US0050A	100	2	$50.00
7	US0100A	100	2	$100.00
8	…….	-	-	-
9	…….	-	-	-
10	…….	-	-	-
11	…….	-	-	-
12	…….	-	-	-
13	…….	-	-	-
14	…….	-	-	-
15	…….	-	-	-
16	…….	-	-	-

The scaling factors will need to be read for all country types in use, and this can be determined by reading all the bill id's first with the 'Request bill id' command and using the first two characters of the return string in the 'Request country scaling factor' command.

19.12. Header 156 - Request country scaling factor

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

The 'scaling factor' and 'decimal places' for any particular country code can be requested by the host machine. If the country code is not recognised then 3 null bytes should be returned by the bill validator i.e. [0] [0] [0].

The scaling factor is used to calculate the monetary value of each bill (typically 1, 100, 1000) and the decimal places is used when displaying this value on the front panel of a machine (typically 0 or 2).

For example, for a $10 bill…

Bill value	Scaling factor	Decimal places	Display as…
10	1	0	$10
10	100	2	$10.00
10	100	0	1000c

This version of the serial protocol allows a typical bill range of…

Bill value	Scaling factor	Decimal places	Display as…
1	1	2	0.01
9999	65535	0	655,284,465

19.13. Header 155 - Request bill position

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

This command is a fast way of determining where bills belonging to particular country are stored. For instance, to enable all Euro bills, the host machine must determine the required inhibit pattern for use with the 'Modify inhibit status' command. This return data can be used directly.

For example, if the return data is [11000000] [00001111] then 6 bills of the specified type are stored in positions 7 to 12.

Bill types 8 to 1	11000000
Bill type 16 to 9	00001111

Sending this data back with the 'Modify inhibit status' command will immediately enable them and inhibit the remainder.

19.14. Header 154 - Route bill

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

[route code]
0 - return bill
1 - send bill to cashbox / stacker

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

This command allows a bill held in the escrow to be routed in one of two ways - back to the customer or into the cashbox or stacker.

Stacking a bill typically takes 2 to 3 seconds. An event will be generated when a bill is returned or stacked.

Note that if the stacker is full then the unit will not accept any bills. Likewise if there is a fault with the stacker. Attempting to route the bill under these conditions returns an error code rather than an ACK.

19.15. Header 213 - Request option flags

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

[feature bit mask]
B0 - stacker supported
B1 - escrow supported
B2 - individual bill accept counters supported
B3 - individual error counters supported
B4 - all counters are non-volatile
B5 - bill teach supported
B6 - bill security tuning supported
B7 - remote bill programming supported

A bit value of 0 (clear) means the feature is not supported. A bit value of 1 (set) means the feature is supported.

This command helps the host software to determine what features are provided by a particular bill validator.

The following cctalk commands are governed by feature support…

B0 - stacker supported
Modify bill operating mode
Request bill operating mode
Perform stacker cycle

B1 - escrow supported
Modify bill operating mode
Request bill operating mode
Route bill

B2 - individual bill accept counters supported
Request individual accept counter

B3 - individual error counters supported
Request individual error counter

B4 - all counters are non-volatile
Host machine does not need to store counter values

B5 - bill teach supported
Teach mode control
Request teach status

B6 - bill security tuning supported
Modify security setting
Request security setting

B7 - remote bill programming supported
Upload bill tables
Begin bill table upgrade
Finish bill table upgrade

19.16. Header 153 - Modify bill operating mode

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

[mode control mask]
B0 - stacker (0 = do not use, 1 = use)
B1 - escrow (0 = do not use, 1 = use)

The host machine can determine whether a stacker and / or escrow are used. It may be the case that on a particular model changing these settings has no effect.

The default mode on power-up is product specific. Changes are RAM-based and volatile.

19.17. Header 152 - Request bill operating mode

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

See above.

19.18. Header 226 - Request insertion counter

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

Counter of bills inserted, regardless of outcome.

This counter may or may not be volatile - check with a specific manufacturer. If it is volatile, then it is up to the host machine to retain audit data between sessions.

19.19. Header 225 - Request accept counter

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

Counter of bills sent to cashbox or stacker.

This counter may or may not be volatile - check with a specific manufacturer. If it is volatile, then it is up to the host machine to retain audit data between sessions.

19.20. Header 150 - Request individual accept counter

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

Each type of bill can have a corresponding accept counter. The value of that counter may be read with this command.

Use the 'Request option flags' command to see if this feature is supported.

This counter may or may not be volatile - check with a specific manufacturer. If it is volatile, then it is up to the host machine to retain audit data between sessions.

19.21. Header 149 - Request individual error counter

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

[error type]
2 - Invalid bill (due to validation fail)
3 - Invalid bill (due to transport problem)
8 - Bill pulled backwards
9 - Bill tamper

See Appendix C (BNV Event Codes) for other error types.

Some types of error can have a corresponding counter. The value of that counter may be read with this command. If the error type is not recognised then the bill validator should not reply.

Use the 'Request option flags' command to see if this feature is supported.

This counter may or may not be volatile - check with a specific manufacturer. If it is volatile, then it is up to the host machine to retain audit data between sessions.

19.22. Header 247 - Request variable set

Transmitted data : <none>
Received data : [variable 1] [variable 2]

Two variables are currently defined, but more may be returned in future.

[variable 1]
No. of bill types concurrently supported (typically 16)

[variable 2]
No. of banks supported (typically 1)

19.23. Header 249 - Request polling priority

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

Refer to the generic specification. A typical bill validator may request a credit polling time of 200ms.

See the section 'Security Features and Typical Operation' for a discussion on polling requirements.

19.24. Header 202 - Teach mode control

Transmitted data : [bill type] [orientation]
Received data : ACK

[orientation]
1 to 4

This command allows a new bill to be inserted into a bill validator a number of times in order to allow the 'pattern' to be recognised in future.

Bills will need to be inserted in a variety of orientations - details of the process have yet to be published. Using this command allows the host machine (via the front console) to put the bill validator into 'teach mode' whilst this process is carried out.

Use the 'Request option flags' command to see if this feature is supported.

19.25. Header 201 - Request teach status

Transmitted data : [mode]
Received data : [no. of bills entered] [status code]

[mode]
0 - default
1 - abort validator teach mode

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

Use the 'Request option flags' command to see if this feature is supported.

19.26. Header 196 - Request creation date

Also known as ‘Date of manufacture’ or ‘Factory set-up date’.

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

Standard cctalk RTBY format (‘relative-to-base-year’).

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

As an example, if the base year was 2000, then a date code of 0021 hex would indicate 1st January, 2000.

The base year can be determined with the ‘Request base year’ command.

19.27. Header 195 - Request last modification date

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

Standard cctalk RTBY format (‘relative-to-base-year’). See previous command.

This command can be used to determine the date the product was last modified e.g. to put in new bill tables.

19.28. Header 170 - Request base year

Transmitted data : <none>
Received data : 4 x ASCII chars e.g. “2000”

The base year of the product in ASCII - usually the year the product was designed. All date codes are relative to this year.

19.29. Header 216 - Request data storage availability

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

Some bill validators may provide a small storage space in memory for use by the host machine (logging machine id numbers etc).

Refer to the generic specification for an explanation of memory types and the block notation.

If no data storage is provided by the bill validator then null bytes should be returned…
[0] [0] [0] [0] [0] - no data storage available

19.30. Header 215 - Read data block

Transmitted data : [block number]
Received data : [data 1] [data 2] [data 3] ...

The read mechanism for user data blocks.

19.31. Header 214 - Write data block

Transmitted data : [block number] [data 1] [data 2] [data 3] ...
Received data : ACK

The write mechanism for user data blocks.

19.32. Header 145 - Request currency revision

Transmitted data : <none> or [country char 1] [country char 2]
Received data : ASCII string

A manufacturer-specific currency revision string is returned. This string can be used to check whether remote programming a new bill set with the 'Upload bill tables' command is necessary.

Certain manufacturers may choose to assign different revision codes to different countries and so the revision string for a specific country can be requested as shown.

If this country code is not supported then the string "Unknown" should be returned..

19.33. Header 144 - Upload bill tables

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

[block]
0 to 255

[line]
0 to 255

This is a generic command to transfer new currency information into a bill validator in a 'manufacturer neutral' format. There is no reference to bill type - it is assumed that this is represented internally within the data structure.

The data is split into block / line segments with 256 lines in each block and up to 128 data bytes in each line. Therefore this command is used repeatedly to transfer all the required currency information.

Any number of bytes between 1 and 128 may be transferred with this command as the message packet already contains a length field.

It is assumed the 'line' counter will be nested within the 'block' counter as follows…

```
for ( block = 0; block <= max_block; ++block )
      {
      for ( line = 0; line <= 255; ++line )
            Send command( block, line, data_array );
      }
```

The maximum data that can be transferred with this command is…
256 x 256 x 128 = 8 Mbytes

If a manufacturer chooses to integrate the firmware and bill tables then this command can be used to transfer both in one go.

To check the currency revision before and after upgrading, use the 'Request currency revision' command.

19.34. Header 143 - Begin bill table upgrade

Transmitted data : <none>
Received data : ACK

This command notifies the bill validator that a currency upgrade is about to take place. The 'Upload bill tables' command can then be used to transfer the information which will be in a manufacturer-specific format.

19.35. Header 142 - Finish bill table upgrade

Transmitted data : <none>
Received data : ACK

This command notifies the bill validator that a currency upgrade has finished.

19.36. Header 141 - Request firmware upgrade capability

Transmitted data : <none>
Received data : [firmware options]

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

To check the firmware revision before and after upgrading, use the 'Request software revision' command.

19.37. Header 140 - Upload firmware

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

This is a generic command to transfer new firmware into a bill validator.

The format is the same as the 'Upload bill tables' command.

19.38. Header 139 - Begin firmware upgrade

Transmitted data : <none>
Received data : ACK

This command notifies the bill validator that a firmware upgrade is about to take place. The 'Upload firmware' command can then be used to transfer the information.

19.39. Header 138 - Finish firmware upgrade

Transmitted data : <none>
Received data : ACK

This command notifies the bill validator that a firmware upgrade has finished.

19.40. Header 137 - Switch encryption code

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

This command allows the encryption security code used in the cctalk encryption layer to be changed 'on-the-fly' by the host machine. Use of this command is optional but it can help increase security by causing seemingly random variations in the same message packet.

'sec 1' to 'sec 6' are 4-bit BCD digits in the range 0 to 9

For example, to change the security code to '123456' we would send [21] [43] [65] in hex to the host machine.

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

The new security code is stored in volatile memory only. To make it permanent then the host machine can send the '' command.

19.41. Header 136 - Store encryption code

Transmitted data : <none>
Received data : ACK

This command makes a change to the encryption security code permanent by having the bill validator write the new value to NV memory. This is part of the encryption technique described more fully in the accompanying security document.

20. Diagnostic Commands in Detail

Note that diagnostic commands are intended for use during power-up initialisation or for 'bench' diagnostic equipment. Access to diagnostic mode may also be given through the host menu system which is usually PIN number or key protected. Diagnostic commands **should not be issued during normal operation** as they could interfere with the bill validation system - particularly the bill transport mechanism.

20.1. Header 232 - Perform self-check

Transmitted data : <none>
Received data : [fault code]

or

Transmitted data : <none>
Received data : [fault code] [extra info]

[fault code]
0 - OK
1 - EEPROM checksum corrupted
See Appendix D (BNV Fault Codes) for the rest.

[extra info]
May be used to report which sensor, motor etc. has a fault.

The fault code table is shared with coin acceptors as there is some overlap with potential fault conditions.

Only one fault can be reported at a time and these are internally prioritised by the bill validator. Fixing one particular fault may reveal another !

20.2. Header 237 - Read input lines

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

This command may be used to read DIP switches etc. for use by diagnostic software.

Manufacturer and product specific.

20.3. Header 238 - Test output lines

Transmitted data : [bit mask]
Received data : ACK

[bit mask]
Each bit :
0 = output off
1 = output on

This command may be used to output test patterns on parallel-mode connectors for use by diagnostic software.

The outputs are typically pulsed for 50ms.

Manufacturer and product specific.

20.4. Header 233 - Latch output lines

Transmitted data : [bit mask]
Received data : ACK

[bit mask]
Each bit :
0 = output off
1 = output on

This command may be used to output test patterns on parallel-mode connectors for use by diagnostic software.

Outputs are latched until cancelled by sending a null byte.

Manufacturer and product specific.

20.5. Header 151 - Test lamps

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

[lamp no.]
Lamp no. 1 to 255

[lamp control]
0 - automatic mode, allow bill validator to control lamp (default)
1 - manual mode, force lamp off
2 - manual mode, force lamp on

Some bill validators have front panel lamps which may be controlled via the host machine. If the host machine has a problem, it could decide to inhibit the bill validator and turn off the front-panel lamps.

If a single lamp is used then the only available lamp number is 1 - other lamp numbers will have no effect.

Automatic mode allows the bill validator to control its own lamps. Inhibiting all the bills in automatic mode would typically cause the bill validator to switch its own lamps off to avoid unwanted attention.

20.6. Header 236 - Read opto states

Transmitted data : <none>
Received data : [opto states]

[opto states]
Each bit :
0 - opto clear
1 - opto blocked

Bit positions are manufacturer and product specific.

20.7. Header 148 - Read opto voltages

Transmitted data : <none>
Received data : [opto voltage 1] [opto voltage 2] [opto voltage 3] …

This command can be used to read various photo-electronic voltages used in the bill validation process. All voltage measurements are scaled into a 8-bit range. Therefore a 5V range could be resolved to an accuracy of about 20mV.

Count	Voltage
255	5.00
192	3.76
128	2.51
64	1.25
0	0.00

Manufacturer and product specific.

20.8. Header 147 - Perform stacker cycle

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

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

This command executes 1 cycle of the stacker for diagnostic purposes. The ACK is only returned after the stacker cycle is completed - typically after a few seconds. An error code may possibly be returned.

20.9. Header 239 - Operate motors

Transmitted data : [motor bit mask]
Received data : ACK

[motor bit mask]
B0 - motor 1 (0 = off, 1 = on)
B1 - motor 2 (0 = off, 1 = on)
B2 - motor 3 (0 = off, 1 = on)
…

This is a legacy command with no control over speed and direction. See below if you require these advanced features.

Manufacturer and product specific. It may not be implemented as a 'public' command for security reasons.

20.10. Header 146 - Operate bi-directional motors

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

This command allows up to 8 motors to be operated in either forward or reverse direction.

[motor bit mask]
B0 - motor 1 (0 = off, 1 = on)
B1 - motor 2 (0 = off, 1 = on)
B2 - motor 3 (0 = off, 1 = on)
…

[direction flags]
B0 - motor 1 (0 = backwards, 1 = forwards)
B1 - motor 2 (0 = backwards, 1 = forwards)
B2 - motor 3 (0 = backwards, 1 = forwards)
…

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

The main bill transport motor will be controlled by 'B0'. If a motor has speed control then it can make use of the 3rd data byte. Otherwise, it will be ignored.

Manufacturer and product specific. It may not be implemented as a 'public' command for security reasons.

21. Switching and Storing Encryption Codes

The ability of the protocol to obscure what is happening from an external attack is dependent on the frequency with which the encryption code is changed. Given sufficient data capture over a long period of time, it would be possible to break any encryption algorithm. However, this task is massively complicated if the code which controls the encryption process is changed on-the fly. As the command to change the encryption code to a new one is itself encrypted, the cryptanalysis process is extremely hard.

It is suggested that the host machine issues a new security code with the 'Switch encryption code' command before every credit poll using the 'Read buffered bill events' command. This could typically be every 200ms.

Storing the encryption code does not need to be done as often and is really for retention of the last code between sessions (when the machine is switched off). This prevents an easy way in for a hacker as there would be no indication of the next power-up code. Check with the manufacturer of the BNV for recommendations on storing encryption codes and any limitations there may be.

Changing the encryption code from the existing one to the next one is obviously a critical period in the operation of the BNV. If power is lost to the device in the middle of the switch then the expected encryption code may not work. It is suggested therefore that the host machine retains the following codes in the event of this type of failure…

- new encryption code
- existing encryption code
- power-up encryption code (the last stored value)
- product label encryption code (entered into the machine)

If a credit poll fails after switching the encryption code to a new one, it is suggested the one prior to switching is tried next, then the power-up code and finally the product label code. The power-up code will be the one in force when the last 'Store encryption code' command was used and the product label code was the one entered into the machine when installing the device. If none of these work then the encryption code has probably been scrambled and so you will need to consult the BNV manufacturer for any possible recovery mechanism.

22. Appendix A - CRC Checksum Algorithm

22.1. 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 ]
```

22.2. Algorithm in C++

```
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;

}
```

```
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;

}
```

22.3. Loop-up Table

```
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
};
```

22.4. 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
```

23. Appendix B - Encryption Example

In this example, we will assume the 6 digit security code is ‘123456’ and the encryption algorithm is something simple - XORing the message bytes with each security code digit in turn (1 then 2 then 3…). The actual encryption algorithm used on bill validators will obviously be far more sophisticated and is not published here.

Taking as a start point the CRC message packets from Appendix A, we apply the encryption algorithm to all bytes after the first two (the destination address and the no. of data bytes).

```
TX : [ 40 ] [ 0 ] [ 70 xor 1 ] [ 1 xor 2 ] [ 63 xor 3 ]
RX : [  1 ] [ 0 ] [ 48 xor 1 ] [ 0 xor 2 ] [ 55 xor 3 ]
```

Applying the XOR function…

```
TX : [ 40 ] [ 0 ] [ 71 ] [ 3 ] [ 60 ]
RX : [  1 ] [ 0 ] [ 49 ] [ 2 ] [ 52 ]
```

This new message is the cctalk ‘Reset device’ command with a CRC checksum and encryption enabled.

24. Appendix C - BNV Event Codes

Bank Note Validator 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

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

25. Appendix D - BNV Fault Codes

Bank Note Validator Fault Codes

Gaps in the table below have already been allocated to coin acceptors and are likely to be specific to that technology.

Code	Fault	Optional Extra Info
0	OK (no fault detected)	-
1	EEPROM checksum corrupted	-
30	ROM checksum mismatch	-
36	Fault on bill validation sensor	Identify which sensor
37	Fault on bill transport motor	-
38	Fault on stacker	-
39	Bill jammed	-
40	RAM test fail	-
41	Fault on string sensor	-
255	Unspecified fault code	-

A single byte of extra information may be returned to locate the fault more precisely. This can be ignored by the host machine if so desired.

26. Appendix E - Manufacturer ID Strings

Some products return full names in response to the 'Request manufacturer id' command, others return abbreviated names. Bill validators will return abbreviated names by convention.

26.1. Full Names

Full Name
Coin Controls Ltd
Money Controls

26.2. Abbreviated Names

Manufacturer	Abbreviated Name
AstroSystems Ltd	AST
Innovative Technology Ltd	ITL
Japan Cash Machine	JCM
Money Controls International	MCI
Mars Electronics International	MEI
National Rejectors Inc	NRI

27. Appendix F - Common Country Codes

Each country of the world has a 2 letter designator which generally follows the convention for car number plates away from home. Listed below are the larger countries, split into 'Europe' and 'Rest of the World'. If you require a code for a country not shown below then please contact Money Controls for a full, up-to-date list.

27.1. Europe

Albania	AL
Austria	AT
Belgium	BE
Bosnia	BA
Bulgaria	BG
Croatia	HR
Cyprus	CY
Czech Republic	CZ
Denmark	DK
Estonia	EE
Euro	EU
Finland	FI
France	FR
Germany	DE
Greece	GR
Hungary	HU
Iceland	IS
Irish Republic	IE
Israel	IL
Italy	IT
Netherlands	NL
Latvia	LV
Lithuania	LT
Luxembourg	BE
Norway	NO
Poland	PL
Portugal	PT
Romania	RO
San Marino	SM
Serbia	SX
Slovakia	SK
Slovenia	SL
Spain	ES
Sweden	SE
Switzerland	CH
Turkey	TR
United Kingdom	GB

27.2. Rest of the World

Algeria	DZ
Argentina	AR
Australia	AU
Bahrain	BH
Bolivia	BO
Brazil	BR
Canada	CA
Chile	CL
China	CN
Columbia	CO
Egypt	EG
India	IN
Indonesia	ID
Iran	IR
Iraq	IQ
Japan	JP
Kenya	KE
Korea North	KP
Korea South	KR
Kuwait	KW
Laos	LA
Latvia	LV
Lebanon	LB
Liberia	LR
Libya	LY
Malaysia	MY
Mexico	MX
Morocco	MA
Mozambique	MZ
Nepal	NP
New Zealand	NZ
Nicaragua	NI
Nigeria	NG
Pakistan	PK
Paraguay	PY
Peru	PE
Philippines	PH
Russia	RU
Saudi Arabia	SA
Senegal	SN
Singapore	SG
South Africa	ZA
Sri Lanka	LK
Sudan	SO
Syria	SY
Thailand	TH
Taiwan	TW
Tunisia	TN
Uganda	UG
Ukraine	UA
United States	US
Uruguay	UY
Venezuela	VE
Vietnam	VN
Yemen	YE
Zaire	ZR
Zambia	ZM
Zimbabwe	ZW

28. Appendix G - Glossary

BNV	Bank Note Validator. The industry standard notation for a bill validator or bill acceptor.
CRC	Cyclic Redundancy Check. A fancy checksum capable of detecting many different types of bit errors.
MDCES	Multi-Drop Command Extension Set. These are the cctalk commands for determining and resolving address ambiguities on a multi-drop network.