Serial Compact Hopper Mk 2

Product Manual

Version 2.7

This document is the copyright of Money Controls Ltd and may not be reproduced in part or in total by any means, electronic or otherwise, without the written permission of Money Controls Ltd. Money Controls Ltd does not accept liability for any errors or omissions contained within this document. Money Controls Ltd shall not incur any penalties arising out of the adherence to, interpretation of, or reliance on, this standard. Money Controls Ltd will provide full support for this product when used as described within this document. Use in applications not covered or outside the scope of this document may not be supported. Money Controls Ltd. reserves the right to amend, improve or change the product referred to within this document or the document itself at any time.

©Money Controls 2005. All rights reserved.

Revision History

Issue	**Date**	**Comments**
1.0	01/08/00	Working draft
1.1	04/10/00	First release
1.2	31/10/00	Removal of 'Opto blocked permanently during payout' test restriction. More clarification on the use of the hardware reset pin. External reset has 10K pull-up.
1.3	30/11/00	Software reset time 'TSinit' increased from 20ms to 40ms
1.4	16/01/01	Clarification of 'Emergency stop' command… 'Power Fail Limitations' section deleted and incorporated into 'Power Fail in Detail' section.
1.5	24/01/01	'Address random' - default coin acceptor address is 2 not 3 The value of the 'coins unpaid' counter is now discussed in much more detail and some examples are given.
1.6	08/05/01	More information on [rx bytes ignored] Change to Opto Security Timing Addition of… Request hopper status : Coding Recommendations
1.7	16/10/01	Addition of Appendix F : Examples of hopper status
1.8	24/06/02	Appendix D : Use cctalk command 168 to read the hopper dispense count Note : Software address changes are volatile.
1.9	05/09/02	Note on [rx bad checksums] counter after sending an encrypted cctalk command to another peripheral.
2.0	29/07/03	Appendix E - mention of polarisation key Appendix F expanded Reference to slow rise time in the 'Power Fail in Detail' section. Appendix G added for the 'unencrypted' version of this product. Request variable set : [connector address] is updated continuously.
2.1	10/10/03	'Modify variable set' command. Single coin mode : any other value is treated as **single coin** payout mode
2.2	04/12/03	Recovery time specified for 'Emergency stop' command. See Appendix B.
2.3	31/03/04	Note on inter-command timing. Addition of Appendix H : Report on 'The Action of an Unexpected Hardware Reset'.
2.4	15/06/04	Appendix F partial result fixed in issue 2.4 software
2.5	01/07/04	Typo in 'Header 163 : Test hopper'. Returns 2 bytes… [hopper status register 1] [hopper status register 2] Appendix F : Change to only one work-around.
2.6	05/08/04	Typo in Simple Coin Payout corrected. Dispense hopper coins command has 9 data bytes.
2.7	18/05/05	TSP number added and released.

©Money Controls 2005. All rights reserved.

Contents

Introduction 4
Why Serial ? 4
Product Features 4
cctalk Design Parameters 6
Serial Connector Type 6
Serial Connector Pinout 7
Auxiliary Connector Type 7
Auxiliary Connector Pinout 8
Address Selection 8
Encryption Mechanism 9
PIN Number Mechanism 10
Power Fail Recovery 12
Power Fail in Detail 12
What happens after Power Up ? 14
What happens after Hardware Reset ? 15
What happens after Software Reset ? 15
Simple Coin Payout 16
Full Initialisation & Payout Sequence 17
Command List 20
Commands in Detail 21
Power Distribution on a Multi-Drop Bus 35
Electrical Noise - Physical Measures 36
Electrical Noise - Software Measures 36
Physical Specification 37
    Current 37
    External Reset 37
    Payout Rate 37
    Environment 37
    Maintenance Schedule 37
    Life 37

Conversion Equations & Default Values Appendix A - 38
Timing Parameters Appendix B - 39
cctalk Interface Circuit Appendix C - 40
NV Memory Map Description Appendix D - 41
Mk2 versus Mk1 Appendix E - 43
Request Hopper Status Examples Appendix F - 44
The Unencrypted Hopper Appendix G - 46
Report on 'The Action of an Unexpected Hardware Reset' Appendix H - 47

©Money Controls 2005. All rights reserved.

Introduction

The Serial Compact Hopper Mk2 or SCH2 is a serially controlled version of the popular Compact Hopper manufactured by Money Controls and an updated version of the original 'Serial Compact Hopper', now known as the 'Mk1' version. The serial interface is **cctalk**, firmly established as a leading, low-speed, device control protocol in the money-transaction industry. A key feature of cctalk is its optimal balance between simplicity and security.

Why Serial ?

Coin hoppers traditionally have a simple parallel interface. Common methods for paying out coins include 'logic motor control' whereby a low voltage control signal can be used to turn the motor on and off, and 'pulse counting' whereby a stream of pulses is used to dispense the coins (one coin per pulse). In the 'logic motor control' method it is up to the host software to monitor and count coins travelling past the payout optos.

Security for the parallel interface method is poor for a determined hacker. Simply drilling a hole in the cabinet and applying suitable voltages to the interface connector or cables can be used to empty the hopper of coins. A hopper with a serial interface is usually resistant to this means of attack due to the sheer complexity of the 'dispense coins' instruction.

Another benefit of 'multi-drop' serial is the ability to connect several coin hoppers to the same wiring harness or 'bus'. This greatly simplifies the cabling within a machine as multiple hoppers can be daisy-chained together rather than having to branch out from a central star point. The number of control signals is usually much less with serial than with parallel. The only control signal in the cctalk protocol is a single bi-directional 'data' line. It is also possible with serial to connect in other money transaction peripherals such as coin acceptors, bill validators and card readers.

The inherent 'expandability' of serial allows for a much better level of diagnostics and error reporting than is available on parallel, if it is available at all. Rather than a general alarm condition, the difference between a coin jam and a deliberate attempt to fraud the hopper can be reported externally.

Product Features

SCH2 represents to date the most sophisticated serial coin hopper in the world. Responding to comments from the leading UK machine manufacturers, Money Controls has designed in an unprecedented level of security.

The following features are available on SCH2…

- Coin dispensing security. The serial command to pay out a coin uses a 64-bit encryption key which changes randomly after each operation. This makes serial 'cloning' whereby a handheld terminal listens to the serial bus and plays back dispense commands a fruitless exercise. A typical dispense cycle will see a total of 192 bits of random information transferred across the bus.

©Money Controls 2005. All rights reserved.

- Payout modes. The hopper defaults to multi-coin payout mode which pays up to 255 coins in a single dispense command. For extra security the hopper can be placed in single coin payout mode which only allows one coin to be paid out at a time.
- PIN number security. There is an option to protect the hopper with a PIN number so that if it is stolen from a machine and plugged into another one it cannot be made to work. It is the equivalent of a mechanical lock with the machine manufacturer generating and keeping the key.
- Opto security. During idling (no coins being paid out), the exit optos are randomly pulsed. If a blockage is seen while driving the opto or a short-circuit seen while not driving the opto then an alarm condition is generated. During pay out, if a short-circuit is seen while not driving the opto then an alarm is generated. Blockages are usually caused by inserting objects into the hopper exit and short-circuits by shining torches or lasers into the optics.
- Motor terminal protection. A heavy duty mechanical relay protects the motor terminals by shorting them out during idle. Any attempt to pay out coins by placing a 9V or 12V battery across the motor terminals will merely short out the battery.
- Anti-jam operation. If the hopper experiences a coin jam during a payout sequence it will automatically reverse in order to clear the jam.
- Software fuse. If an absolute maximum current threshold is exceeded (factory pre-set) then the hopper aborts payout with an error code.
- Polyswitch protection. The motor driver terminals are protected with a polyswitch for additional overload protection.
- Power fail protection. A non-volatile memory keeps track of coins paid out. If power is lost during a payout sequence then the residual number of coins to pay can be read back after the machine re-initialises.
- Unique serial number. Each hopper is manufactured with a unique 24-bit serial number which cannot be modified by external means.
- Coin counting. Two counters record the number of coins paid out of the hopper. One is reset-able by the user, the other is a life counter. Both are implemented in NV Memory.
- Data integrity. All coin counter values in NV Memory are stored with a 8-bit checksum to ensure data integrity.
- Level plate support. There is an option to fit high or low level plates and the status of these can be read by the host machine on serial.
- Remote configuration. Motor parameters such as reversing current and payout timeout can be changed with serial commands. No changes to electronic components are required.
- Multi-drop operation. A number of serial hoppers can be connected to the same serial bus. Device addresses default to those determined by the wiring harness but they can be changed in software to any 8-bit value.
- User memory. 10 bytes of non-volatile memory are available for unrestricted use by the host machine. There are various security and auditing tasks which could be accomplished with this feature.
- Extensive command set. Host software can implement a small or large fraction of the full command set available depending on the application. Commands are available for inhibiting the hopper, reading the state of the exit optos, checking the software revision etc.

©Money Controls 2005. All rights reserved.

- Diagnostic and error reporting. Full access to diagnostic and error codes are made available over serial.
- Code protection. The software is protected with an internal, independent, watchdog circuit. A 'crash' in the software will result in a clean reset of code.

cctalk Design Parameters

Refer to issue 3.2 of the 'cctalk Serial Communication Protocol / Generic Specification' for an explanation of the protocol and its implementation on any platform.

This product is configured as…
cctalk b96.p0.v24.a5.d0.c8.m0.x8.i1.r3

In other words…
9600 baud / open-collector interface / +24V supply / +5V data /
supply sink / connector type 8 / slave device / 8-bit checksum /
implementation level 1 / spec. issue 3

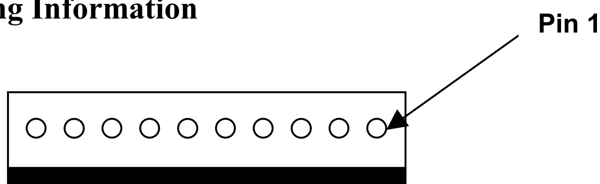
The hopper can **only** operate at 9600 baud.

Serial Connector Type

PCB Connector
2.54mm (0.1inch) pitch with locking wall

Part No. : TBD

Keying Information



View of Connector from Top

©Money Controls 2005. All rights reserved.

Serial Connector Pinout

Pin	Function
1	Address select 3 - MSB
2	Address select 2
3	Address select 1 - LSB
4	+Vs
5	+Vs
6	0V
7	0V
8	/DATA (cctalk)
9	N/C
10	/RESET

Operation can be achieved with just 3 wires…
- +24V to pin 4
- GND to pin 6
- Bi-directional serial data line to pin 8

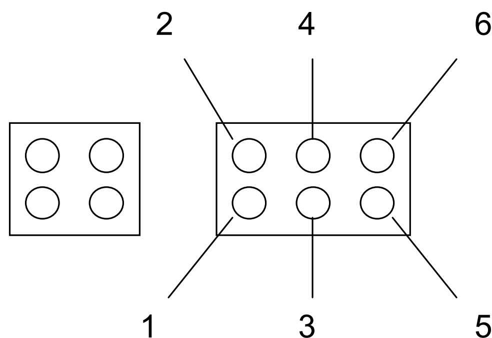
Pins 4 and 5, and pins 6 and 7, are linked internally. The provision of extra pins is to simplify the manufacture of a multi-drop cable using thicker wire for the power leads. There can be a 'power-in' and a 'power-out' pin, and the hoppers daisy-chained..

Auxiliary Connector Type

PCB Connector
2.54mm (0.1inch) pitch

Part No. : TBD

Keying Information



©Money Controls 2005. All rights reserved.

Auxiliary Connector Pinout

Pin	Function
1	High Level Plate
3	Low Level Plate
5	Plate Common
2	High Level Link
4	Low Level Link
6	Link Common

Operation

To notify the hopper software that level plate sensors are fitted, the link pins should be connected as follows…

Mode	Connections
High level plates only	pin 2 to pin 4
Low level plates only	pin 4 to pin 6
High & low level plates	pin 2 to pin 4 to pin 6

Otherwise, no connections should be made.

The level plates themselves should be connected through the corresponding plate pin (pin 1 for high level, pin 3 for low level) and the plate common (pin 5).

Address Selection

The default cctalk bus address for a 'Payout' device is 3. This is the address of the Serial Compact Hopper if no connections are made to the address select pins (pins 1 to 3) on the connector.

For applications requiring more than one hopper on the serial bus, one or more of the address select lines may be connected to +Vs. A total of 8 unique bus address may be generated in this way, in the range 3 to 10 inclusive.

X = Connect to +Vs (Pins 4, 5)			Serial Address
Address select 3	Address select 2	Address select 1	
			3
		X	4
	X		5
	X	X	6
X			7
X		X	8
X	X		9
X	X	X	10

A number of mating connectors on a multi-drop bus cable may each be wired uniquely to allow operation of multiple hoppers. Since address selection is done

©Money Controls 2005. All rights reserved.

externally, any Serial Compact Hopper may be plugged into any position on the bus and the host machine will know which one is paying out a particular coin.

☞Address determination from the connector is only done at power-up or reset. Changing the address select lines afterwards has no effect.

Note : Addresses may be changed in software to values other than those in the above table. Refer to the 'Address change' and 'Address random' serial commands. These values are lost at power-down or reset.

Encryption Mechanism

A 64-bit encryption mechanism is used to ensure that an illegal attempt to dispense coins from SCH2 is a hugely difficult task. The key to this mechanism is a secret algorithm, not published in this document, which may be obtained from Money Controls after suitable approval procedures have been gone through.

To show the procedure for dispensing a coin an example is shown here with a 'trivial' encryption mechanism but the overall procedure is the same. Byte values between brackets are shown in hex.

First of all we pump the random number generator of the hopper by sending 8 bytes of random data to it…

Command = Pump RNG
Transmitted data : [34] [A2] [D7] [0F] [35] [17] [55] [94]
Received data : ACK

This is not an essential step but is useful to broaden the spectrum of cipher keys that are transmitted along the serial bus and which may be 'recorded' by a hacker. As the host machine is likely to be an AWP machine with a sophisticated random number generator, way beyond the capability of the hopper microcontroller, we may as well make use of it. Note that the pump value does not pre-set or 'seed' the RNG as that would defeat the security mechanism, but only scrambles it further. The exact details of the scrambling algorithm will not be documented.

Then we request a key cipher key…

Command = Request cipher key
Transmitted data : <none>
Received data : [E5] [88] [13] [07] [46] [FE] [29] [05]

A new cipher key must always be requested prior to dispensing coins. There is no point using an old copy as it changes after every dispense command. The 'Request cipher key' command itself can be repeated in the event of a communication error and the cipher key will be re-transmitted rather than regenerated.

Now we combine the cipher key with the number of coins to pay out (in this example 20 coins or the value 14 hex) by tagging it onto the end of the data block…

©Money Controls 2005. All rights reserved.

Non-encrypted data = [E5] [88] [13] [07] [46] [FE] [29] [05] [14]

In this case we will assume that the CMF (Cryptographic Mapping Function) is simply inverting all the bytes (new data = FF - old data).

Performing this calculation on each of the bytes we obtain…

Encrypted data = [1A] [77] [EC] [F8] [B9] [01] [D6] [FA] [14]

Note that the number of coins to pay out is unencrypted but its value is used in the real CMF.

Now we send that data to the hopper to pay out a coin…

Command = Dispense hopper coins
Transmitted data : [1A] [77] [EC] [F8] [B9] [01] [D6] [FA] [14]
Received data : [5] - example event counter

The next time we pay out a coin the cipher key will have changed, so unless the algorithm is known, simple command 'cloning' will not work.

In practice, the CMF will be far more complex than this example and will require considerable computing power and a long observation period to 'crack'. Firing random data at the hopper will also prove fruitless as there are astronomical odds against a successful code match.

Money Controls is realistic enough to appreciate that eventually the CMF may fall into the wrong hands whether though huge computing resource or through leaked documentation. We have embedded into our system an undocumented mechanism for changing the CMF in response to future industry needs without any change to hardware.

An obvious hack would be to intercept and change the number of coins dispensed by the 'Dispense hopper coins' command to a greater value. This has been taken care of in the security algorithm and will not work.

PIN Number Mechanism

A PIN number is provided on SCH2 as an **optional** security feature. By default, units are shipped without the PIN number mechanism enabled. If this feature is not required or its use is too restrictive then it can simply be ignored.

By programming a PIN number into the device, if the hopper device is subsequently powered down or removed to another location then unless the PIN number is known, no coins can be dispensed. This is another layer of defence against the determined hacker who wishes to experiment with the encryption mechanism. However it does require the host machine keeping track of the PIN numbers of any hoppers used in that cabinet.

©Money Controls 2005. All rights reserved.

Various possibilities include…

1. Don't use a PIN number
Nice and easy that one.
2. Fix the PIN number to the same value always
This can be done but is not very secure. Once the PIN number is known then there is effectively no PIN number protection on any of the hoppers. It is simple to manage though and the 'master' PIN number is unlikely to be forgotten.
3. Scramble the PIN number and store in the user memory
This is quite a clever idea because it means you can randomise the PIN number on each hopper and as long as you know how you scrambled it, it can be recovered, unscrambled and sent to the hopper during the initialisation routine. Security relies on keeping this scrambling algorithm secret.
4. Log the PIN number versus serial number
As each hopper has a unique serial number then this gives a convenient method of storing the serial number against a random PIN number in a central database which all the machines have access too on a network. This is the most secure method because unless the PIN number transaction is captured on the bus at just the right moment in time, and for that particular hopper, the only way to obtain the PIN would be by exhaustive searching. With 4.3 billion combinations at 245ms per guess would take on average 16.7 years.

If you are unfortunate enough to have a hopper for which you have forgotten the PIN number then contact Money Controls for details of any possible recovery mechanism that we may have in place at the time.

Example

This is how to set the PIN number on a new hopper to 1-2-3-4 which does not already have one…

Command = Enter new PIN number
Transmitted data : [1] [2] [3] [4]
Received data : ACK

Subsequent use of this command will still return an ACK but will not actually change the PIN number to any other value. This is a 'use once' command.

As soon as a PIN number is programmed, the 'Dispense hopper coins' command will fail until this PIN number is re-entered with the command below.

Likewise, after powering up SCH2 with the PIN number mechanism enabled, it must be entered prior to paying out coins.

Command = Enter PIN number
Transmitted data : [1] [2] [3] [4]
Received data : ACK

Note that an ACK is always returned, even if the PIN number is incorrect. This increases security.

©Money Controls 2005. All rights reserved.

Power Fail Recovery

SCH2 contains a non-volatile memory (EEPROM) for the storage of coin counters. Therefore if power is removed in the middle of a payout sequence then the situation can be recovered and the residual coins paid out after power is restored. This behaviour is under control of the host software - the hopper does not automatically dispense coins when power is re-applied.

The following counters are saved…
- [Last payout : coins paid] x 1 byte
- [Last payout : coins unpaid] x 1 byte
- [Hopper dispense count] x 3 bytes
- [Hopper life dispense count] x 3 bytes

along with their corresponding checksums.

The [Last payout : coins paid] and [Last payout : coins unpaid] bytes can be read with the 'Request hopper status' command.

The [Hopper dispense count] can be read with the 'Request hopper dispense count' command.

The [Hopper life dispense count] can be read by looking at block 3 of the NV Memory using the 'Read data block' command. Refer to Appendix D for a memory map description.

After power-up initialisation, the host machine can read the [Last payout : coins unpaid] byte to determine if there are any remaining coins to be paid out after the last session. The decision to pay out any remaining coins is made by the host machine, not the hopper.

Power Fail in Detail

The sequence of saving the coin counters to NV Memory is triggered by the following conditions…

- Sudden loss of power with the motor running
- Receipt of an 'Emergency stop' command

If power is suddenly lost with the motor running then the hopper will stop the motor immediately and update the NV Memory while it has power to do so. SCH2 does not have a 'battery back-up' but uses a capacitor reservoir.

If power is lost after a payout sequence has completed (hopper in idle) then the [Last payout : coins unpaid] counter is cleared, regardless of the value it was holding. This is because it is assumed the host machine has dealt with the last payout sequence and taken appropriate action. It is not desirable to flag unpaid coins during the next power-up initialisation.

©Money Controls 2005. All rights reserved.

This means a slow rise time or switch bounce on the power supply could inadvertently clear the unpaid counter.

If the host machine has early notification of a power fail it can send an 'Emergency stop' command to the hopper. This command stops the motor dead and returns the number of unpaid coins back to the host. This value **should be stored by the host machine** prior to power being lost.

The use of the hardware reset line (pin 10 of the connector) is not to be used for aborting payout as this does not allow the coin counters to be saved. **The coin counters will be incorrect if the hardware reset pin is used during a payout sequence**.

Examples…

a) Power lost during payout sequence

Counter	Initial value	Pay 3 from 10 then lose power
Last payout : coins paid	0	3
Last payout : coins unpaid	0	→7←
Hopper dispense count	0	3
Hopper life dispense count	N	N + 3

Coins remaining = 7

b) 'Emergency stop' command issued during payout sequence

Counter	Initial value	Pay 3 from 10 then 'Emergency stop'	Cycle power off then on
'Emergency stop' return value	0	→7←	0
Last payout : coins paid	0	3	3
Last payout : coins unpaid	0	7	**ZERO**
Hopper dispense count	0	3	3
Hopper life dispense count	N	N + 3	N + 3

Coins remaining = 7

©Money Controls 2005. All rights reserved.

c) <u>‘Emergency stop’ command issued during payout sequence AND a coin is seen after the hopper replies with unpaid coins</u>

Counter	Initial value	Pay 3 from 10 then ‘Emergency stop’ + **late coin** exit	Cycle power off then on
‘Emergency stop’ return value	0	7 ➔6🡐 on retry	0
Last payout : coins paid	0	4	4
Last payout : coins unpaid	0	➔6🡐	**ZERO**
Hopper dispense count	0	4	4
Hopper life dispense count	N	N + 3	N + 4

Coins remaining = 6

In this more complicated example, the hopper dispense count and the hopper life dispense count end up with the correct values even though a coin was seen on the exit optos after the motor stopped. For the host machine to find the correct value of unpaid coins it would need to re-send the ‘Emergency stop’ command or use the ‘Request hopper status’ command **before** power was lost - otherwise it would think there were 7 unpaid coins rather than 6.

Therefore if you need to know the number of remaining coins during a power fail and wish to use the ‘Emergency stop’ command, please ensure that you have enough time to send this command and a ‘Request hopper status’ before power is lost. This gives the best possible accuracy. The host machine needs at least **100ms** of notice before the power supply dips below Vtrip (see Appendix B).

What happens after Power Up ?

The following is a guide to what happens when power is lost and re-applied.

Device Address
Defaults to the connector address

PIN Number
Retained but needs to be re-entered

Motor Variables
[current limit] = default value
[motor stop delay] = default value
[payout timeout] = default value
[maximum current measured] = ZERO

See Appendix A for default values.

©Money Controls 2005. All rights reserved.

Flags
Refer to the Flag Action Table within the 'Test hopper' command description.

Note that the 'Power-up' flag is set to indicate the power supply really was lost and the hopper defaults to multi-coin payout mode. The hopper also starts inhibited and needs to be enabled prior to coin dispensing.

Counter checksum flags are updated.

Counters
Hopper dispense count = last value
Hopper life dispense count = last value

Request hopper status
[event counter] = ZERO
[payout coins remaining] = ZERO
[last payout : coins paid] = last value
[last payout : coins unpaid] = ZERO or last value if power lost during payout

Request comms status variables
[rx timeouts] = ZERO
[rx bytes ignored] = ZERO
[rx bad checksums] = ZERO

What happens after Hardware Reset ?

The only difference between a hardware reset and a power-up is as follows.

- If the power-up flag has been cleared by a software reset then it is left cleared.

What happens after Software Reset ?

The following is a guide to what happens after a software reset. A software reset means sending the 'Reset device' command to the hopper.

Device Address
Defaults to the connector address

PIN Number
Retained but needs to be re-entered

Motor Variables
[current limit] = default value
[motor stop delay] = default value
[payout timeout] = default value
[maximum current measured] = ZERO

See Appendix A for default values.

Flags

©Money Controls 2005. All rights reserved.

Refer to the Flag Action Table within the 'Test hopper' command description.

Note that the 'Power-up' flag is cleared. The hopper also starts inhibited and needs to be enabled prior to coin dispensing.

Counter checksum flags are updated.

Counters
Hopper dispense count = last value
Hopper life dispense count = last value

Request hopper status
[event counter] = ZERO
[payout coins remaining] = ZERO
[last payout : coins paid] = last value
[last payout : coins unpaid] = last value

Request comms status variables
[rx timeouts] = ZERO
[rx bytes ignored] = ZERO
[rx bad checksums] = ZERO

Simple Coin Payout

The following shows the minimum command set needed to pay out some coins after receiving a 'factory-fresh' hopper. This will be helpful when writing new software drivers or for testing.

We will assume that the hopper is on address 3 and has just powered-up.

Send the following commands…

- Enable hopper
- Request cipher key
- Dispense hopper coins
- Request hopper status

In detail…
Message packets are shown in full (not just the data part). The host address is assumed to be 1 and [byte values] are in decimal.

Enable hopper
TX: [3] [1] [1] [164] [165] [178]
RX: [1] [0] [3] [0] [252] = ACK

©Money Controls 2005. All rights reserved.

Request cipher key
TX: [3] [0] [1] [160] [92]
RX: [1] [8] [3] [0] [key 1] [key 2] [key 3] [key 4] [key 5] [key 6]
[key 7] [key 8] [checksum]

The data bytes returned by this command are used in the encryption algorithm. For security reasons details of the encryption algorithm are not given in this document.

Dispense hopper coins
TX: [3] [9] [1] [167] [sec 1] [sec 2] [sec 3] [sec 4] [sec 5] [sec 6]
[sec 7] [sec 8] [N coins] [checksum]
RX: [1] [1] [3] [0] [event counter] [checksum]

Between 1 and 255 coins can be dispensed with this command.

Request hopper status
TX: [3] [0] [1] [166] [86]
RX: [1] [4] [3] [0] [event counter] [payout coins remaining]
[last payout : coins paid] [last payout : coins unpaid] [checksum]

This command should be repeated until [payout coins remaining] = 0. Another cipher key can then be requested and more coins dispensed.

Full Initialisation & Payout Sequence

This is an example of a full initialisation and payout sequence on SCH2…

Notation : cctalk command’s are shown highlighted.

Optional… **Address Resolution**
If { network indeterminate } then
Address poll ⇨ resolve addresses with BROADCAST address

While { Address clash on any address } then
Address random ⇨ scramble addresses with BROADCAST address
Address poll ⇨ resolve addresses with BROADCAST address

Request equipment category id ⇨ identify each device on bus
Address change ⇨ assign new addresses if required

In other words, if we do not know where the serial hoppers are on the bus then we need to locate them. If any of the device addresses clash then we need to scramble them and look again. However, since the wiring harness for the hopper should initialise the device addresses in a non-ambiguous manner, this address resolution software can be avoided.

©Money Controls 2005. All rights reserved.

Inititialisation
Simple poll ⇨ returns ACK to confirm device is attached, powered-up and 9600 baud comms is working fine
Request equipment category id ⇨ returns 'Payout'. Otherwise you will probably be trying to dispense a coin from an acceptor !
Request variable set ⇨ connector address (physical position in machine)
Request serial number ⇨ store for reference

Optional… **Product Data**
Request manufacturer id ⇨ e.g. 'Money Controls'
Request product code ⇨ e.g. 'SCH2'
Request software revision ⇨ e.g. 'SCH2-Vx.y'
Request comms revision ⇨ cctalk comms level
Request hopper coin ⇨ coin name if stored
Request build code ⇨ level sensor details

PIN Number Unlocking
Test hopper ⇨ check to see if PIN number mechanism is enabled
If { PIN enabled } then
    Enter PIN number
else
    Enter new PIN number ⇨ randomise and log against serial number for future reference
    Optional… Write data block ⇨ log host id in user data memory

The PIN number mechanism should only be enabled if the host machine has some way of remembering it ! The easiest method is to make all the PIN numbers the same but that is not very secure. It is much better to randomise them but in that case there will be problems if hoppers are swapped between machines. You may wish to implement a central database of PIN numbers versus serial numbers or perhaps make use of the user data section of the hopper memory to store an encrypted PIN number.

Product Configuration
Modify variable set ⇨ set current limit
set motor stop delay
set payout timeout
set single coin payout mode

Default values can be used if so desired in which case this command can be skipped. Single coin payout mode is more secure though because once in this mode an external bus attack can at best only pay out a single coin at a time rather than emptying the contents of the hopper. This assumes all the other security mechanisms have been defeated. However, single coin payout mode is much slower.

Remove Payout Inhibit
Enable hopper ⇨ prepare for payout by allowing the dispense command

©Money Controls 2005. All rights reserved.

Check for Residual Payout from a Power Fail
Test hopper ⇨ check for NV Memory write fail or checksum error
If { counter error } then
    Write data block ⇨ if good host copy exists then correct counter
else
    Request hopper status ⇨ check to see if there are residual coins after last payout. If so then take the decision to pay out the balance.

    Optional… Request hopper dispense count ⇨ check with last known host copy to see if an illegal payout has been made

Errors in the hopper memory should not occur under normal operating conditions. If one of the counters become corrupted however it would be possible to restore their contents if the host machine has a backup copy.

Dispense Coin
Test hopper ⇨ check error flags are clear
If { error flags } then
    < take appropriate action on possible fraud attempt >
    Reset device ⇨ clear error flags
    Enter PIN number ⇨ if enabled
    Modify variable set ⇨ set current limit, stop delay, timeout, single coin mode
    Enable hopper ⇨ re-enable hopper after reset

Pump RNG ⇨ send some random numbers to the hopper
Request cipher key ⇨ ready for encryption algorithm
Dispense hopper coins ⇨ pay out one or more coins

If { Dispense NAK } then
    Something went wrong - find out why ?

Dispense Verification
Request hopper status ⇨ check event counter
If { event counter not incremented } then
    Retry Dispense hopper coins

While { payout coins remaining > 0 }
    Request hopper status

Test hopper ⇨ check error flags are clear
If { error flags set } then
    Resolve from the following error conditions…
    <Coin jam> if max. current exceeded
    <Hopper empty> if payout timeout
    <Opto fraud attempt> if any opto error flag

©Money Controls 2005. All rights reserved.

Optional… **Level Auditing**
Request hopper dispense count ⇨ store host copy for subsequent verification
Request payout high / low status ⇨ is the hopper nearly empty ?

Early Notification of Power Fail
Emergency stop
Request hopper status ⇨ store unpaid coins for next power-up initialisation
Wait for power to disappear

Command List

34 commands are supported…

Header 254 : Simple poll
Header 253 : Address poll
Header 252 : Address clash
Header 251 : Address change
Header 250 : Address random
Header 247 : Request variable set
Header 246 : Request manufacturer id
Header 245 : Request equipment category id
Header 244 : Request product code
Header 242 : Request serial number
Header 241 : Request software revision
Header 236 : Read opto states
Header 219 : Enter new PIN number
Header 218 : Enter PIN number
Header 217 : Request payout high / low status
Header 216 : Request data storage availability
Header 215 : Read data block
Header 214 : Write data block
Header 192 : Request build code
Header 172 : Emergency stop
Header 171 : Request hopper coin
Header 169 : Request address mode
Header 168 : Request hopper dispense count
Header 167 : Dispense hopper coins
Header 166 : Request hopper status
Header 165 : Modify variable set
Header 164 : Enable hopper
Header 163 : Test hopper
Header 161 : Pump RNG
Header 160 : Request cipher key
Header 004 : Request comms revision
Header 003 : Clear comms status variables
Header 002 : Request comms status variables
Header 001 : Reset device

©Money Controls 2005. All rights reserved.

Commands in Detail

All byte values shown in [decimal] unless otherwise stated.

Header 254 : Simple poll

Transmitted data : <none>
Received data : ACK

This is a good command to use to confirm that a device is plugged into the expected address, powered-up and operating correctly. A total of 5 bytes are sent to the hopper which then replies with 5 bytes.

Header 253 : Address poll

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

Only a single byte is returned by the hopper rather than a full cctalk message packet.

See the generic specification for more details.

Header 252 : Address clash

Transmitted data : <none>
Received message : {variable delay} <slave address byte>

Only a single byte is returned by the hopper rather than a full cctalk message packet.

See the generic specification for more details.

Header 251 : Address change

Transmitted data : [address]
Received data : ACK

The address specified overrides that determined by the connector wiring loom. The new value is lost at power-down or reset.

Header 250 : Address random

Transmitted data : <none>
Received data : ACK

The address is randomly set to a value between 3 and 255. The broadcast address 0, the default bus master address 1, and the default coin acceptor address 2, are automatically avoided for your convenience. The new value is lost at power-down or reset.

©Money Controls 2005. All rights reserved.

Header 247 : Request variable set

Transmitted data : <none>
Received data : [current limit] [motor stop delay] [payout timeout]
[maximum current measured] [supply voltage]
[connector address]

[current limit]
This is the current threshold at which the motor reverses in order to clear jams.

Refer to CURLMT in Appendix A for details of scaling and its default value.

[motor stop delay]
This is the time the motor is allowed to run on for after detecting the last coin in a payout sequence and should be sufficient for the coin to exit cleanly.

Refer to STOPDLY in Appendix A for details of scaling and its default value.

[payout timeout]
This is the total amount of time allowed for **each** coin to be paid out, allowing for some reversing in the event of a jam. After this time, the motor is stopped.

Refer to PAYTIM in Appendix A for details of scaling and its default value.

[maximum current measured]
Measured with the same units as [current limit]. The current is sampled and averaged as the motor is running and should be used as an approximate guide only.

This measurement can be cleared to zero with the ‘Reset device’ command.

[supply voltage]
The hopper can measure its own power supply voltage and report it back to the host machine. The supply voltage is sampled continuously when not paying out coins.

Refer to SUPVOLTS in Appendix A for details of scaling.

[connector address]
Range 0 to 7.

This is the number specified by the address select pins on the connector. The device address on the cctalk bus is this value plus 3 if it hasn’t been changed subsequently using serial commands.

Note that this value is continuously updated unlike the actual comms address.

Software Design Note : Future products may see some additional information returned by this command. To ensure backwards compatibility, the existing data packet will be retained (in both order and type) and any additional information tagged to the end. If a feature is not supported, the relevant byte will return zero.

©Money Controls 2005. All rights reserved.

Header 246 : Request manufacturer id

Transmitted data : <none>
Received data : "Money Controls"

Header 245 : Request equipment category id

Transmitted data : <none>
Received data : "Payout"

Header 244 : Request product code

Transmitted data : <none>
Received data : "SCH2"

SCH2 = Serial Compact Hopper Mk2.

Header 242 : Request serial number

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

This is a 24-bit binary serial number.

Prototype units return…

```
[ 78 ] [ 97 ] [ 188 ] in decimal
[ 4E ] [ 61 ] [  BC ] in hex
```

= 12,345,678

If you work it out as 5,136,828 you have the bytes in reverse order !

Production units will be supplied with a unique, incremental serial number. Serial numbers cannot be changed by any easy means.

Header 241 : Request software revision

Transmitted data : <none>
Received data : "SCH2-Vx.y"

x, y = 0, 1, 2… depending on the revision level of the software.

©Money Controls 2005. All rights reserved.

Header 236 : Read opto states

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

[payout opto]
Bit mask :
B0 - payout opto A (0 = path clear, 1 = path blocked)
B1 - payout opto B
B2 - payout opto C
B3 to B6 - not used, 0 returned
B7 - payout opto A + B + C

The payout opto whose status is returned on bit 7 is actually comprised of 3 separate opto paths - A, B and C. If a coin is seen at any of these optos then bit 7 is set.

The optos are continuously sampled in the background and the current state is reported by this command.

Note that this command is designed for test purposes only (checking coin visibility on the opto-electronics) and not for counting coins during a payout sequence ! Counting coins during payout is handled automatically by the software and is performed a lot faster than polling serially.

Header 219 : Enter new PIN number

Transmitted data : [PIN1] [PIN2] [PIN3] [PIN4]
Received data : ACK

A factory-fresh hopper has the PIN number mechanism disabled.

A manufacturer can subsequently program the PIN number to any chosen value using this command and after that the number cannot be changed, even if the existing PIN number has been entered correctly. It is a ‘once-only’ lockout mechanism - turning the power off and on does not clear the PIN number.

A PIN number of [0] [0] [0] [0] is legal and would have to be entered.

The ‘Dispense hopper coins’ command is the only one which is ‘blocked’ by the PIN number mechanism - all other commands operate as normal.

Entering a new PIN number after one has already been programmed will still result in the return of an ACK even though the PIN number remains unchanged.

The ‘Test hopper’ command can be used to see if a PIN number has been programmed - refer to bit 7 of ‘hopper status register 2’.

©Money Controls 2005. All rights reserved.

Header 218 : Enter PIN number

Transmitted data : [PIN1] [PIN2] [PIN3] [PIN4]
Received data : ACK

If the PIN number mechanism is enabled then the 'Dispense hopper coins' command will not work unless the correct PIN number has been entered. The current PIN number is lost after a…

- power-down
- hardware reset
- software reset
- emergency stop with the motor running

Incorrect PIN numbers are always ACK'ed as if they had worked.

Header 217 : Request payout high / low status

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

This command returns the status of the level sensor.

[level status]
Bit mask :
Bit 0 - Low level sensor status (1 = lower than low level trigger)
Bit 1 - High level sensor status (1 = higher than or equal to high level trigger)
Bit 2 - not used
Bit 3 - not used
Bit 4 - Low level sensor support (1 = feature supported and fitted)
Bit 5 - High level sensor support (1 = feature supported and fitted)
Bit 6 - not used
Bit 7 - not used

For the operator…
Bit 0 is set if the hopper is NEARLY EMPTY.
Bit 1 is set if the hopper is NEARLY FULL.

The normal operating condition of the hopper is bits 0 & 1 clear.

If the raw level sensor inputs are reading NEARLY EMPTY as well as NEARLY FULL then this is an illegal condition (it can only be one or the other) and bits 0 & 1 are both left clear.

The level sensor inputs are debounced with a time Tlevdeb in Appendix B to remove the effects of shifting coins.

©Money Controls 2005. All rights reserved.

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]

[memory type]
= 2, permanent (limited use)
[read blocks]
= 4
[read bytes per block]
= 8
[write blocks]
= 3
[write bytes per block]
= 8

In other words, 32 bytes of NV Memory are available for reading, 24 bytes for writing, accessed 8 bytes at a time.

Refer to Appendix D for details of the memory map.

Header 215 - Read data block

Transmitted data : [block number]
Received data : [data 1] [data 2]… [data 8]

[block number]
0 to 3

Provides read access to the NV Memory.

Refer to Appendix D for details of the memory map.

Header 214 - Write data block

Transmitted data : [block number] [data 1] [data 2]… [data 8]
Received data : ACK

[block number]
0 to 2

Provides write access to the NV Memory.

Refer to Appendix D for details of the memory map.

©Money Controls 2005. All rights reserved.

Header 192 : Request build code

Transmitted data : <none>
Received data : 8 x ASCII chars

The build code is determined automatically by the hopper during initialisation. It is assumed any build options do not change while power remains on the unit, otherwise a software reset needs to be issued.

```
'Lev HiLo' for high and low level sensor fitted
'Lev Hi  ' for high level sensor only fitted
'Lev   Lo' for low level sensor only fitted
'Standard' for standard model, no extras ( as per SCH1 )
```

Header 172 : Emergency stop

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

If this command is sent during a payout then the motor is stopped immediately as a precursor to power being lost. The returned byte counter should indicate the remaining coins to be paid out.

The effect on the 'Request hopper status' command is as follows…

[payout coins remaining] ⇨ Cleared to ZERO
[last payout : coins paid] ⇨ Coins paid prior to stopping
[last payout : coins unpaid] ⇨ Coins unpaid prior to stopping

The 'Emergency stop' command produces a similar action to a software reset when the motor is running. This means that the hopper will need to be re-initialised (PIN number, enable etc.) prior to dispensing further coins.

When the motor is not running, this command returns the [payout coins remaining] value without performing a reset.

Header 171 : Request hopper coin

Transmitted data : <none>
Received data : 6 x ASCII chars

This command returns an ASCII string consisting of 6 characters. These are stored in the NV Memory.

Coin names are returned as [C] [C] [V] [V] [V] [I] - refer to the cctalk generic specification for an explanation of the format.

The 'unprogrammed' state of the coin name may typically be 6 x ASCII code 45 (------) or 6 x ASCII code 0.

©Money Controls 2005. All rights reserved.

Note : There is no facility currently in production software to program in the name of the coin that will be dispensed. The coins that can be dispensed from a hopper are subject to the physical limitations of the disc / bed assembly rather than any software-controlled parameters. The machine manufacturers will be at liberty to program in their own coin labels using the 'Write data block' command.

Header 169 : Request address mode

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

The address selection method is determined by the product and in this case the value **4A hex** is always returned. The use of this command is 'informational only' but will be useful in future when a number of serial hopper products exist on the market which have different address selection mechanisms.

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

4A hex = Address stored in RAM
Address selection via interface connector
Address may be changed with serial commands (volatile)

The address of the hopper defaults to that in the connector wiring after a power-up or hardware / software reset.

Header 168 : Request hopper dispense count

Transmitted data : <none>
Received data : [no. of coins 1 - LSB] [no. of coins 2] [no. of coins 3 - MSB]

Range 0 to 16,777,215.

The dispense counter records the number of coins dispensed since the counter was last cleared.

The counter returned is the 'reset-able' one, not the life counter which is made available via the 'Read data block' command. See the memory map description in Appendix D for more details and the mechanism for clearing the counter.

Header 167 : Dispense hopper coins

©Money Controls 2005. All rights reserved.

Transmitted data : [sec 1] [sec 2] [sec 3] [sec 4]
[sec 5] [sec 6] [sec 7] [sec 8] [N coins]
Received data : [event counter]

[sec…] security bytes
The 'Dispense hopper coins' command is protected by a sophisticated encryption mechanism. In multi-coin payout mode, between 1 and 255 coins can be dispensed, and the command is transmitted with a **64-bit cryptographic code**. Unless the cryptographic key matches exactly then no coins can be paid out.

If 'N coins' is set to zero then no coins are paid out but the message is handled normally and the event counter is incremented.

[event counter]
Value of the event counter **after** it has been incremented.
Normal range = 1 to 255. Zero indicates a power-up or reset has occurred.

If the received data packet has a checksum error then the event counter cannot be relied upon and this command should not be repeated to avoid over-payment of coins. The event counter must be read with the 'Request hopper status' command and compared with the last known value.

For the payout to occur as intended, **the following conditions have to be met…**

- Valid cctalk message - no errors in format or checksum
- PIN number has previously 'unlocked' the hopper (if the mechanism is enabled)
- Cryptographic code is correct
- No. of coins = 1 in single coin payout mode
- Hopper enabled - see 'Enable hopper' command
- No 'Absolute maximum current exceeded' error flag
- No 'Opto fraud attempt, path blocked during idle' error flag
- No 'Opto fraud attempt, short-circuit during idle' error flag
- No 'Opto fraud attempt, short-circuit during payout' error flag
- No 'Opto blocked permanently during payout' error flag

If any of the above error flags are set (test by using the 'Test hopper' command) then payout is blocked. These flags can be cleared with the 'Reset device' command.

If the payout is blocked for any of the reasons above (apart from a low level comms error which will result in no reply) then a cctalk **NAK message** is returned. The event counter is still incremented and the **next encryption key generated by the hopper**.

The reason for a dispense fail is deliberately not returned for security reasons.

©Money Controls 2005. All rights reserved.

Header 166 : Request hopper status

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

[event counter]
Every valid 'Dispense hopper coins' command increments the event counter. Valid means there was no comms error in the command packet.

The event counter only has the value 0 at power-up or reset - if its value is 255 and another dispense command is received then the event counter changes to 1. Then 2, 3, 4 etc.

The event counter is added for security reasons - if the reply to a 'Dispense hopper coins' command is corrupted or missing due to noise then the event counter should be checked prior to re-transmitting the command to prevent over payout of coins. Correctly written host software should always check the event counter before re-sending a dispense command.

[payout coins remaining]
After a 'Dispense hopper coins' command this counter is primed with the number of coins to pay out. It then **decrements** with each coin dispensed until it reaches zero.

[last payout : coins paid]
The number of coins paid out in the last 'Dispense hopper coins' command. This counter **increments** as coins are being paid out.

[last payout : coins unpaid]
The number of coins which failed to be paid out in the last 'Dispense hopper coins' command. This counter is **cleared** during a payout.

As soon as a dispense hopper coins command is received, the status bytes are updated as follows…

[payout coins remaining] ⇨ 'N coins' / **decrements** as each coin is paid out
[last payout : coins paid] ⇨ Cleared to ZERO / **increments** as each coin is paid out
[last payout : coins unpaid] ⇨ Cleared to ZERO

When payout is completed (success or abort) then the status bytes become…
[payout coins remaining] ⇨ Cleared to ZERO
[last payout : coins paid] ⇨ Correct value for last operation
[last payout : coins unpaid] ⇨ Correct value for last operation

Host software should always wait for [payout coins remaining] to reach zero before deciding what to do next.

©Money Controls 2005. All rights reserved.

Request hopper status : Coding Recommendations

Using life test results, Money Controls can now make the following coding recommendations…

Polling the hopper status after a 'Dispense hopper coins' command is best done every **200ms**. It is essential that a retry mechanism is put in place such that if there is no reply to the 'Request hopper status' command after **50ms** then it is sent again. The number of **sequential** retries allowed before a hardware fault is suspected should be set at around **10**. The reason for the heavy retry mechanism is the electrical noise generated by the motor. If suitable measures are taken in software then the serial communication link should be 100% reliable and this has been confirmed by tests at Money Controls.

Header 165 : Modify variable set

Transmitted data : [current limit] [motor stop delay]
[payout timeout] [single coin mode]
Received data : ACK

Refer to 'Request variable set' for more details - the format of data is the same.

[single coin mode]
0 - set multi-coin payout mode (default)
1 - set single coin payout mode
(any other value is treated as single coin payout mode)

For security reasons, once a hopper is set to single coin payout mode it cannot be changed back to multi-coin payout mode with this command. A power-down or hardware / software reset is required.

This command allows some of the motor control variables to be modified but does not necessarily have to be sent before using the hopper. The default values listed in Appendix A are normally optimal.

Variable set changes are volatile. Any custom values are lost at power-down or reset.

Software Design Note : Future products may see some additional information sent by this command. To ensure backwards compatibility, the existing data packet will be retained (in both order and type) and any additional information tagged to the end. If a feature is not supported, the relevant byte will be ignored.

©Money Controls 2005. All rights reserved.

Header 164 : Enable hopper

Transmitted data : [enable code]
Received data : ACK

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

The hopper must be enabled before coins can be paid out. Only the decimal value 165 (A5 hex, 10100101 binary) enables it, all other values disable it.

If a hopper is disabled prior to a 'Dispense hopper coins' command, a NAK is returned.

A 'Reset device' command or a power-down cycle will cause the hopper to be disabled by default.

This command is retained for compatibility with SCH1 rather than its negligible effect on security.

Header 163 : Test hopper

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

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

[hopper status register 1]
Bit mask :

B0 - Absolute maximum current exceeded	1 = exceeded
B1 - Payout timeout occurred	1 = occurred
B2 - Motor reversed during last payout to clear a jam	1 = reversed
B3 - Opto fraud attempt, path blocked during idle	1 = fraud
B4 - Opto fraud attempt, short-circuit during idle	1 = fraud
B5 - Opto blocked permanently during payout	1 = blocked
B6 - Power-up detected	1 = power-up
B7 - Payout disabled	1 = disabled

The 'Payout timeout occurred' flag is cleared prior to each dispense operation.

Once it has been set, the 'Motor reversed during last payout to clear a jam' flag stays latched through subsequent dispense operations but may be cleared with a software reset.

©Money Controls 2005. All rights reserved.

[hopper status register 2]
Bit mask :

B0 - Opto fraud attempt, short-circuit during payout	1 = fraud
B1 - Single coin payout mode	1 = single
B2 - Checksum A error	1 = error
B3 - Checksum B error	1 = error
B4 - Checksum C error	1 = error
B5 - Checksum D error	1 = error
B6 - Power fail during NV Memory write	1 = fail
B7 - PIN number mechanism	1 = enabled

'Opto fraud attempt, short-circuit during payout' occurs if light is shone at the hopper exit optos during a dispense operation.

Refer to Appendix D for details of the checksum types.

Flag Action Table

Short Label	Dispense coins requirement ?	Action on Power-Up or Hardware Reset	Action on Software Reset
Max. Curr.	Yes	Cleared	Cleared
Timeout	-	Cleared	Cleared
Reverse	-	Cleared	Cleared
Idle block	Yes	Cleared	Cleared
Idle s/c	Yes	Cleared	Cleared
Block	Yes	Cleared	Cleared
Power-up	-	Set on PU only	Cleared
Disabled	Yes	Set	Set
Pay s/c	Yes	Cleared	Cleared
Single	-	Cleared	Cleared
Check A	-	Calculated	Calculated
Check B	-	Calculated	Calculated
Check C	-	Calculated	Calculated
Check D	-	Calculated	Calculated
NV Fail	-	'NV Memory'	'NV Memory'
PIN	-	'NV Memory'	'NV Memory'

Header 161 : Pump RNG

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

This command pumps the random number generator of the hopper with extra random variables to make prediction of the next random number in the sequence a lot harder. Its use is optional but recommended where security is paramount.

©Money Controls 2005. All rights reserved.

Header 160 : Request cipher key

Transmitted data : <none>
Received data : [key 1] [key 2] [key 3] [key 4]
[key 5] [key 6] [key 7] [key 8]

This command requests the encryption key required for coin payout. It may be requested repeatedly in the event of a comms error since it only changes after a…

- ‘Dispense hopper coins’ command
- ‘Pump RNG’ command
- Power-down or any type of reset

Header 004 : Request comms revision

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

[cctalk level]
= 1
[major revision]
= 3
[minor revision]
= 2

In other words, the first issue level of cctalk specification 3.2

Header 003 : Clear comms status variables

Transmitted data : <none>
Received data : ACK

Clears the counters returned by the ‘Request comms status variables’ command.

Header 002 : Request comms status variables

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

[rx timeouts]
Number of receive message timeouts recorded by the processor.

There is an inter-byte timeout value of Trxout - see Appendix B. The cctalk protocol has a variable length packet structure but data timeouts can be detected and counted.

[rx bytes ignored]
Number of receive bytes ignored by the processor (due to receive buffer overflow).

SCH2 has a receive buffer length of 15 bytes for the data part of the packet - the header and checksum are stored elsewhere. Packets with more than 15 bytes of data

©Money Controls 2005. All rights reserved.

increase the ignored counter by the **total** number of data bytes. So sending a message with 20 data bytes would increase the ignored bytes by 20. Likewise a message with 15 or less would not do anything to the counter, a message with 16 by 16 and a message with 252 by 252.

[rx bad checksums]
Number of messages received by the processor with bad checksums.

☞ Note that if an encrypted command is sent on a multi-drop bus (e.g. to a bill validator) then the hopper records this as a checksum fail as it verifies the checksum prior to an address match, rather than afterwards.

All these counters are cumulative and wrap around to 0 after 255.

When testing a new software driver, it is worth checking these counters after a series of message transactions to confirm all is well.

Header 001 : Reset device

Transmitted data : <none>
Received data : ACK

This is the command required for a 'software reset'.

An ACK is returned **prior** to resetting.

Host software should allow a delay after the ACK before sending the next command to allow the hopper initialisation code to complete - see TSinit in Appendix B.

After a 'Reset device' command, various status flags are cleared (see the 'Test hopper' command).

Any motor parameters sent with a 'Modify variable set' command will return to their default values. Default values are given in Appendix A.

Power Distribution on a Multi-Drop Bus

The multi-drop bus for Serial Compact Hoppers consists of a power, ground and serial data line. When more than one hopper is attached to the bus, all power is transferred along a single cable and significant ground potential rises can occur.

The following recommendations are made…

- Only operate one serial hopper at a time. Never initiate a second payout sequence when one is already in progress.
- Consider the use of signal conditioning on the serial data line receiver. Perhaps a high-frequency filter and voltage comparitor input with a mid-rail (2.5V) threshold.
- Consider running separate power cables to the hoppers to alleviate the ground potential problem.

©Money Controls 2005. All rights reserved.

- If communication errors still occur, consider changing the topology of the multi-drop bus network. A star network will distribute power more evenly than a ring, tree or daisy-chain network.

Electrical Noise - Physical Measures

The cctalk protocol is not designed for long distance transfer but for local hook-up of various peripherals within a machine cabinet. Typical cable lengths are likely to be of the order of a few metres.

Various measures can be taken to minimise the effects of radiated and conducted noise on the cctalk bus.

- Use a good quality regulated power supply with mains filtering. The power rating should be sufficient to handle a Serial Compact Hopper at maximum surge current.
- Do not run the multi-drop bus cables directly next to noisy electrical components if at all possible. These are typically motors, relays, VDU's, fluorescent strip lights etc. If problems are experienced consider the use of screened cable.
- Keep cable runs as short as possible.
- Make sure the cctalk data line has an appropriate load resistor at the host end (typically 1K to 10K pull-up to +5V).
- Do not place too many peripherals on the bus - consider the loading effects of each cctalk interface circuit. The maximum number allowed will depend on the host transceiver circuit.

Electrical Noise - Software Measures

There is a big difference in security and reliability terms between a good software implementation of cctalk and that of a poor one.

The following design points should be noted carefully…

Check each cctalk reply packet for errors.
- Was there a low level byte framing error ?
- Were there the correct number of data bytes in the message ?
- Was the return destination address correct ?
- Was the checksum correct ?
- Was the peripheral source address the one expected ?
- Was the return header zero ?

If a reply is returned with an error then the cctalk command can be re-transmitted as many times as deemed appropriate - this is a key feature of cctalk.

Note that the [rx bad checksums] byte returned by the 'Request comms status variables' command is useful for monitoring noise.

©Money Controls 2005. All rights reserved.

Physical Specifciation

Current

24 volt motor version

Voltage	19 to 26V dc	
Current	when empty	300mA typical
	when full	1.0A typical
	at switch-on	3.0A peak
	on reversing	3.5A peak

External Reset

Signal Active low with 10K pull-up to 5V
Input volts, low 0.6V max
Input volts, high 3.5V min (5.0V max)

Payout Rate

Rate, multi-coin payout mode	8 to 10 coins/s typical
Rate, single-coin payout mode	2 coins/s approx.

The much slower single-coin payout mode is due mainly to the overheads of encryption.

Environment

Operating temperature	0 to 60°C
Storage temperature	-20 to 70°C
Operating humidity	10 to 75% RH
Storage humidity	10 to 95% RH, non-condensing

Maintenance Schedule

Every 100,000 coins	clean light guide with a damp cloth
Every 500,000 coins	replace ejector fingers
Every 1,000,000 coins	replace adjuster plate

Life

Expected product lifetime 3 million coins with routine maintenance

©Money Controls 2005. All rights reserved.

Appendix A

Conversion Equations & Default Values

Label	Scaling & Units	Default Byte Value	Default Physical Value
CURLMT	N / 17.1 Amps	34	2.0A
STOPDLY	N ms	0	0ms
PAYTIM	N * 0.333 s	30	10s
SUPVOLTS	0.2 + N * 0.127 Volts	188	24V

Limits

The value of CURLMT cannot be set lower than 6 (which corresponds to 0.35A) otherwise the unit will not operate properly. If a value smaller than 6 is specified, the original value is retained.

The value of STOPDLY cannot be set greater than 50 (50ms) for security reasons. If a value larger than 50 is specified, the value is clamped at 50.

Software Reset

A software reset will force the motor parameters to return to their default values, regardless of their current state.

©Money Controls 2005. All rights reserved.

Appendix B

Timing Parameters

Nominal values are shown.

Label	Description	Value	Units
Xtal	PIC Resonator	4.0	MHz
Brate	Serial communication baud rate	9600	baud
Trxout	Receive data timeout	25	ms
Vtrip	Power-fail trip threshold (+24V motor)	16	V
TPinit	Power-up initialisation time	600	ms
TRinit	Hardware reset initialisation time	20	ms
TSinit	Software reset initialisation time	< 40	ms
Tstop	Emergency stop recovery time (note α)	< 250	ms
TMRelOn	Motor-on relay delay	25	ms
TMRelOff	Motor-off relay delay	25	ms
TMreverse	Motor reversing time	150	ms
Irelay	Motor protection relay rating (surge)	10	A
TEESave	EEPROM data save time	17	ms
Ifuse	Absolute maximum trip current	5	A
Tlevdeb	Level sensor debounce time	2	s
	Comms reply delay : Simple poll	< 2	ms
	Comms reply delay : Enter PIN number (correct)	< 2	ms
	Comms reply delay : Enter PIN number (wrong)	235	ms
	Comms reply delay : Write data block	8	ms
	Comms reply delay : Dispense hopper coins	< 4	ms
	Comms reply delay : Request hopper status	< 2	ms
	Comms reply delay : Pump RNG	< 2	ms
	Comms reply delay : Request cipher key	< 2	ms

Note α : Only when motor is running. No delay is needed if not dispensing.

Opto Security Timing

When idle, the hopper software tests the optos on a pseudo-random basis to make a deliberate fraud attempt much harder. The sampling period varies between 15 and 255 milliseconds. When paying out coins, the sampling period changes to between 64 and 127 milliseconds.

Hopper Initialisation

No serial communication is possible during the ‘TPinit’, ‘TRinit’ and ‘TSinit’ time.

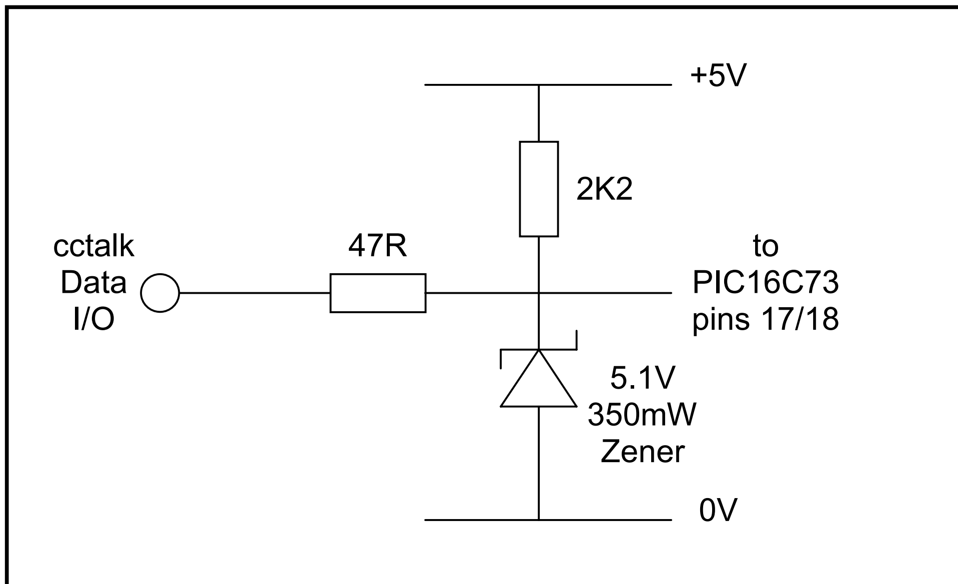
Inter-command Timing

It is recommended that there is a gap of at least 1ms between receiving a hopper reply and sending the next command.

©Money Controls 2005. All rights reserved.

Appendix C

cctalk Interface Circuit



This is the cctalk electronic interface circuit on SCH2.

There are many options for the host interface circuit but we recommend an open-collector drive.

©Money Controls 2005. All rights reserved.

Appendix D

NV Memory Map Description

Block No.	Length Bytes	Description	Read / Write Permission
0	8	User data	R / W
1	6	Coin name	R / W
1	2	User data	R / W
2	3	Hopper dispense count	R / W
2	1	Checksum A	R / W
2	1	Last payout : coins paid	R / W
2	1	Checksum B	R / W
2	1	Last payout : coins unpaid	R / W
2	1	Checksum C	R / W
3	3	Hopper life dispense count	Read Only
3	1	Checksum D	Read Only
3	1	Black Box Recorder A	Read Only
3	1	Black Box Recorder B	Read Only
3	1	Black Box Recorder C	Read Only
3	1	Black Box Recorder D	Read Only

User data
10 bytes of user data are available to the host machine for any kind of storage requirement. The data can be ASCII test or binary data - there is no restriction on format. All 8 bytes of block 0 are available, together with the top 2 bytes of block 1.

Coin name
The name of the hopper coin, if programmed. Users can program their own coin names. The ASCII string stored in these locations is returned by the 'Request hopper coin' command.

Hopper dispense count
The total no. of coins dispensed since the counter was reset.

☞ Note that the cctalk command 'Request hopper dispense count' should be used to read the value of this counter. The value in EEPROM is not updated until power-down or reset.

Last payout : coins paid
The no. of coins dispensed by the last command.

Last payout : coins unpaid
The no. of coins which failed to be dispensed by the last command.

Hopper life dispense count
The total no. of coins dispensed since the product was manufactured.

©Money Controls 2005. All rights reserved.

☞ Note that the life dispense count is **only updated** when the NV Memory is updated - i.e. when power is removed or at reset. It does not hold a running count value like the 'hopper dispense count'.

Checksums and Black Box Recorders
Each counter is protected by a null checksum such that the 8-bit addition of all counter bytes and the checksum byte is zero.

If the software detects a checksum fail then the corresponding checksum error flag is set (refer to the 'Test hopper' command) and the black box recorder byte is incremented. The black box recorder bytes are used to monitor problems with the power-fail electronics.

Example - Clearing the Hopper dispense count

This can be achieved by reading block 2 and writing it back with the appropriate bytes cleared to zero.

Complete message packets shown. Data values in decimal. Hopper is address 3.

Read Cycle
TX : [3] [1] [1] [215] [2] [checksum] - Read block 2
RX : [1] [8] [3] [0] [count 1] [count 2] [count 3] [count checksum]
[data 1] [data 2] [data 3] [data 4] [checksum]

Write Cycle
TX : [3] [9] [1] [214] [2] [0] [0] [0] [0] - Write block 2
[data 1] [data 2] [data 3] [data 4] [checksum]
RX : [1] [0] [3] [0] [252] - ACK

©Money Controls 2005. All rights reserved.

Appendix E

Mk2 versus Mk1

If you already have a machine interface for the Mk1 version of the Serial Compact Hopper then the following information may be of interest.

The following changes have been made on the Mk2 version…

- Serial connector has 2 extra pins at the end - pins 1 to 8 are the same. The mechanical polarisation key has also changed sides.
- The 'Dispense hopper coins' command is radically different - it requires an 8 byte crypto code and returns the new event counter rather than an ACK
- The 'Test hopper' command now returns 2 bytes of flags rather than 1 byte
- The 'Modify variable set' command takes an extra byte
- Various product identification strings have changed

This is the command set difference between Mk2 and Mk1…

cctalk Command	Supported on Mk2 ?	Supported on Mk1 ?
Header 254 : Simple poll	X	X
Header 253 : Address poll	X	X
Header 252 : Address clash	X	X
Header 251 : Address change	X	X
Header 250 : Address random	X	X
Header 247 : Request variable set	X	X
Header 246 : Request manufacturer id	X	X
Header 245 : Request equipment category id	X	X
Header 244 : Request product code	X	X
Header 242 : Request serial number	X	X
Header 241 : Request software revision	X	X
Header 236 : Read opto states	X	X
Header 219 : Enter new PIN number	X	
Header 218 : Enter PIN number	X	
Header 217 : Request payout high / low status	X	X
Header 216 : Request data storage availability	X	X
Header 215 : Read data block	X	
Header 214 : Write data block	X	
Header 192 : Request build code	X	X
Header 172 : Emergency stop	X	X
Header 171 : Request hopper coin	X	X
Header 169 : Request address mode	X	X
Header 168 : Request hopper dispense count	X	X
Header 167 : Dispense hopper coins	Note α	X
Header 166 : Request hopper status	X	X
Header 165 : Modify variable set	Note α	X
Header 164 : Enable hopper	X	X
Header 163 : Test hopper	Note α	X
Header 161 : Pump RNG - new header	X	
Header 160 : Request cipher key - new header	X	
Header 004 : Request comms revision	X	X
Header 003 : Clear comms status variables	X	X
Header 002 : Request comms status variables	X	X
Header 001 : Reset device	X	X

Note α : The data packets have been modified on these commands

©Money Controls 2005. All rights reserved.

Appendix F

Request Hopper Status Examples

The reply from the 'Request hopper status' command is…

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

Dispense 1 coin in single coin mode

```
Hopper status = 01 01 00 00
Hopper status = 01 01 00 00
Hopper status = 01 01 00 00
Hopper status = 01 00 01 00 Paid 1 coin
```

Dispense 5 coins in multi-coin mode

```
Hopper status = 02 05 00 00
Hopper status = 02 05 00 00
Hopper status = 02 04 01 00
Hopper status = 02 04 01 00
Hopper status = 02 03 02 00
Hopper status = 02 03 02 00
Hopper status = 02 02 03 00
Hopper status = 02 02 03 00
Hopper status = 02 01 04 00
Hopper status = 02 01 04 00
Hopper status = 02 00 05 00 Paid 5 coins
```

Dispense 5 coins in multi-coin mode
Only 2 coins in hopper

```
Hopper status = 03 05 00 00
Hopper status = 03 04 01 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 03 02 00
Hopper status = 03 01 01 03 Partial result - ignore
Hopper status = 03 00 02 03 Paid 2 coins, 3 remaining
```

©Money Controls 2005. All rights reserved.

Dispense 10 coins in multi-coin mode
Only 3 coins in hopper

```
Hopper status = 01 0A 00 00
Hopper status = 01 09 01 00
Hopper status = 01 08 02 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 01 02 07 Partial result - ignore
Hopper status = 01 00 03 07 Paid 3 coins, 7 remaining
```

The partial result takes the following form…
[payout coins remaining] ➔ 1
[last payout : coins paid] ➔ 1 less than final value
[last payout : coins unpaid] ➔ correct value

If host machine software is displaying the number of coins paid then the best work-around is to show [last payout : coins paid] ONLY if the current value is greater than the last value. This will 'hide' the partial result problem.

Note from June 2004 onwards…

The partial result problem has been fixed in issue 2.4 of the software and above. To see which version of software your hopper has, send cctalk command header 241, 'Request software version'.

In the above example…

```
Hopper status = 01 07 03 00
Hopper status = 01 07 03 00
Hopper status = 01 00 03 07 Paid 3 coins, 7 remaining
```

©Money Controls 2005. All rights reserved.

Appendix G

The Unencrypted Hopper

Serial Compact Hopper Mk2 is available in an unencrypted version.

The unencrypted serial compact hopper can be used without any knowledge of the encryption algorithm used on the standard 'encrypted' product. The cctalk commands have been kept identical to minimise changes to software between products.

To find out which product you have, use cctalk header 244, 'Request product code'.

Product	Encryption ?	Returned Product Code
Serial Compact Hopper	Yes	SCH2
Serial Compact Hopper	No	SCH2-NOENCRYPT

When the 'Dispense hopper coins' command is sent, 8 zeros can be sent rather than the encryption code.

Note that a 'Request cipher key' command is still required prior to each dispense but the result can be thrown away as it is not needed. If the request cipher key is not sent then a NAK will be returned when a dispense is attempted.

Example Command Sequence

1. Command 'Enable hopper'
```
TX = 003 001 001 164 165 178
RX = 001 000 003 000 252 = ACK
```

2. Command 'Request cipher key'
```
TX = 003 000 001 160 092
RX = 001 008 003 000 097 162 057 066 112 230 076 011 201
```

The 8 data bytes can be ignored as they are not needed.

3. Command 'Dispense hopper coins'
```
TX = 003 009 001 167 000 000 000 000 000 000 000 000 005 071
RX = 001 001 003 000 008 243
```

This examples dispenses 5 coins.

The 8 encryption codes are ignored and can be set to zero as in this example.

©Money Controls 2005. All rights reserved.

Appendix H

Report on 'The Action of an Unexpected Hardware Reset'

Introduction

Like any electronic device, serial hoppers can suffer from unexpected interruptions as a result of, for example, external interference, fraud attempts etc. One such event is an unexpected hardware reset. The consequence of this reset will vary from machine to machine depending on how the controller software deals with the problem. This bulletin discusses the issues arising out of such a reset and puts forwards various solutions.

The serial hopper can be instructed to pay out a certain number of coins and it will report back, through host polling, when the payout is complete along with confirmation of the coins paid and the coins unpaid. If power to the hopper fails during a payout sequence, a non-volatile memory stores where the hopper is up to so that on restoration of power the remaining coins can be paid out. The problem with a hardware reset during payout (which causes the motor to brake immediately) is that no action can be taken by the hopper microcontroller to store any data so it is like the hopper is powering up with no knowledge of the previous payout operation. The serial hoppers only store coin count values on detection of a power-down or brown-out and also on the issue of a software reset or emergency stop command.

See 'Example Comms Log' below for an example of the reset problem.

Solutions

Host software can be written in a number of ways, each one trading simplicity with security. The simplest by far is the 'fire and forget' mode where the host machine instructs the hopper to dispense say 30 coins and then waits until the hopper reports back it has finished (through host polling). A far more secure approach would be to track the coins going through the hopper in host software and look for any discrepancies if and when they arise.

Here are some solutions for consideration which would provide robustness against unexpected hardware resets in the hopper.

(1) Write secure controller software. If a reset occurs, the next valid status command will return an event counter of zero (if the reset occurred in the middle of a message then there could be a comms error of some kind - no reply, bad checksum, incorrect length etc.). When a 'Test hopper' command is then sent, it will be seen that the hopper is disabled but no power-up event has occurred - the classic signature of a hardware reset. Once a reset is detected, the contents of the coins paid and coins unpaid counters from the hopper should be discarded as they do not apply to this payout event. It is therefore inadvisable to resume payout on the basis of the remaining coins as indicated by the hopper. If the termination of payout was due to a power fail then this could be done, but not for a reset. So the choice then is to go into a 'service callout' mode or to resume payout on the basis of host-tracked coins. The

©Money Controls 2005. All rights reserved.

host machine knows the coins paid at each polling event so worst case the host would not know about coins from a poll to a reset just prior to the next poll. If we assume 10 coins per second worst case in multi-coin mode (stream mode) then an additional coin could be dispensed every 100ms. If the polling interval is faster than 100ms then there is unlikely to be more than a 1 coin error.

(2) Dispense small batches of coins only. If 1, 2 or 3 coins are paid out at a time, the maximum 'liability' on receipt of a sudden reset will be no more than this number. For maximum security, the mark 2 serial hopper can be placed in single coin payout mode (pulse mode) using a serial command. In this mode, the hopper will only accept dispense commands for 1 coin at a time. For single coin payout, each coin is dispensed and acknowledged in software before the next coin is paid out. Although this payout method is much slower (2 to 3 coins per second), it is a lot more secure.

Conclusion

In terms of a 'good practice' software and operating guide…

- Ensure the host software keeps track of the coins dispensed as the hopper is polled by maintaining a running 'coins remaining' counter independently of the hopper.

- When the coins paid counter reaches zero, check the event counter for a zero and test the hopper to see if a reset has occurred. If it has, do not use the hopper paid and unpaid counters as they will be wrong. Also the hopper will need to be enabled prior to the next payout.

- Payout can resume after a reset subject to an occasional, small, overpay.

- Poll the hopper at least every 100ms for maximum security.

- If speed is not an issue then dispense coins in single coin mode or in small batches of no more than 3 coins at a time in multi-coin mode.

©Money Controls 2005. All rights reserved.

Example Comms Log

An example of a comms log from a hopper which has suddenly reset is shown below. It was sent the command to dispense 250 coins in stream mode.

Picking up the log after 65 coins have been paid out…

```
(195)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 185 065 000 253 - OK
From Device  : Return message header
```

Payout event 1, 185 coins remaining.

```
(196)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 185 065 000 253 - OK
From Device  : Return message header

(197)
    +37ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 184 066 000 253 - OK
From Device  : Return message header
```

Payout event 1, 184 coins remaining.

```
(198)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 184 066 000 253 - OK
From Device  : Return message header

(199)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 184 066 000 253 - OK
From Device  : Return message header

(200)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 183 067 000 253 - OK
From Device  : Return message header
```

Payout event 1, 183 coins remaining.

```
(201)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 183 067 000 253 - OK
From Device  : Return message header
```

©Money Controls 2005. All rights reserved.

```
(202)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 183 067 000 253 - OK
From Device  : Return message header

(203)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 182 068 000 253 - OK
From Device  : Return message header
```

Payout event 1, 182 coins remaining.

```
(204)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 182 068 000 253 - OK
From Device  : Return message header

(205)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 182 068 000 253 - OK
From Device  : Return message header

(206)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 181 069 000 253 - OK
From Device  : Return message header
```

Payout event 1, 181 coins remaining.

```
(207)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 181 069 000 253 - OK
From Device  : Return message header

(208)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 180 070 000 253 - OK
From Device  : Return message header
```

Payout event 1, 180 coins remaining.

```
(209)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 180 070 000 253 - OK
From Device  : Return message header

(210)
    +35ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 180 070 000 253 - OK
From Device  : Return message header
```

©Money Controls 2005. All rights reserved.

```
(211)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 001 180 070 000 253 - OK
From Device  : Return message header

(212)
    +36ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
*** NO REPLY ***
```

Comms error !

```
(213)
   +526ms - 003 000 001 166 086 - OK
From Machine : Request hopper status
     +0ms - 001 004 003 000 000 000 095 155 254 - OK
From Device  : Return message header
```

Payout event 0, 0 coins remaining.

Hopper reports paid 95 coins, 155 coins remaining.
Hopper actually paid 70 coins, 180 coins remaining.

Potential underpay of 25 coins.

```
(214)
    +39ms - 003 000 001 163 089 - OK
From Machine : Test hopper
     +0ms - 001 002 003 000 128 000 122 - OK
From Device  : Return message header
```

Flag status = [128] [0] = Hopper disabled

So in this hopper dump, the hopper has stopped after paying out 70 coins. It reports back it is has paid out 95 coins which is clearly incorrect. The hopper is also disabled without a disable command being sent. We know power has not failed because bit 6 of the first status byte is not set.

The coins remaining was 180 and this suddenly jumps to 155 coins remaining which is suspicious. The final reported 'coins unpaid' value after a hardware reset may be higher or lower than the actual value - it depends on the last value saved to non-volatile memory. Relying on this value to resume payout after a reset would result in either an overpay or an underpay. In the example above it is an underpay of 25 coins.

©Money Controls 2005. All rights reserved.