SSP SMILEY ® SECURE PROTOCOL

INTELLIGENCE IN VALIDATION

Innovative Technology
INTELLIGENCE IN VALIDATION

Automated Transactions
INTELLIGENCE IN VALIDATION

Bellis Technology
INTELLIGENCE IN VALIDATION

CHANGE HISTORY

Innovative Technology Ltd

Title:	SSP Gaming Protocol		
Drawing No:	GA138	**Project:**	
Author:	P. Dunlop	**Date:**	26/05/98
Format:	MS Word 2000		

Issue	Protocol Ver	Release Date	Comments
Issue 1	1	26/05/98	
Issue 2	1	03/02/99	
Issue 3	1	11/06/99	
Issue 4	2	4/02/00	
Issue 5	2	20/06/00	
Issue 6	2	26/10/00	
Issue 7	2	20/11/00	
Issue 8	2	20/01/01	
Issue 9	2	4/10/01	
Issue 10	2	21/01/02	
Issue 11	2	23/03/04	General Revision
Issue 12	3	05/08/04	Protocol Version 3
Issue 13	3	02/01/06	Protocol Version 3 events
Issue 14	4	16/11/07	BNV Barcode commands
Issue 14L	4		Encryption, Smart Range
Issue 14M	4	21/6/09	Encryption, Smart Range
Issue 15	4	23/7/09	Encryption, Smart Range Release
Issue 16	4	1/9/09	Payout command mod
Issue 17	4	30/11/09	New coin mech events fro SMART HOPPER
Issue 18	6	16/02/10	Protocol Version 6
Issue 19	6	04/05/10	Add Note Float commands
Issue 20	6	10/05/10	
Issue 21	6	27/05/10	
Issue 22	6		
Issue 23	6		
Issue 24	6	21/10/10	
Issue 25	7	01/11/10	Additional v7 commands

Document Issue 25 - Protocol Version 7

Issue 4. – 13/01/2000
Introduction of generic commands. Introduction of commands/ responses for coin readers and coin hoppers. Introduction of addressing structure. Introduction of encrypted packets. Upgrade "Protocol Version' to 2.

Issue 5 – 20/06/2000
Clarification of remote download specification – addition of example sequences. Correction of command conflict – SYNC and LAST REJECT CODE specified with same code, LAST REJECT CODE has been changed to 0x17. Addition of notes to identify function not currently implemented on NV4 / NV4X.

Issue 6. – 26/10/00
Block size missing from header description in version 5 – fixed. Rewording of description of remote downloading protocol. Addition of maximum block size. No code changes required in product firmware or demo code. Changed example CRC code from assembler example to C example. Addition of simplified remote programming flow chart.

Issue 7. – 20/11/00
Introduction of basic card reader commands. Addition of FAIL as a generic response. Addition of manufacturers extension generic command for use internally.

Issue 8. – 20/01/01
SLAVE_RESET form generic response to an event response to reflect correct behaviour. Addition of euro county code to appendix.

Issue 9. – 04/10/01
Addition of HOLD command to allow escrow implementation on BNV

Issue 10 – 21/01/02
Addition of slave address for a Audit Collection Device

Issue 11 – 23/03/04
General Revision Correction of Slave ID Reference in section 3.1
Addition of extra address allocation for note validators

Issue 12 – 05/08/04
Addition of SHOW_RESET_EVENTS BNV commands and note cleared at reset events. Protocol taken to Version 3

Issue 13 – 02/01/06
Addition of Cash box removed and replaced events. Explanation of Note start-up events in expanded protocol.

Issue 14 – 16/11/07
The addition of Bar code ticket commands and events for Banknote validator. Updated reject reason codes.

Issue 14K – 07/5/2009

Revision of Encryption layer to propose a more secure system & interface for smart hoppers and smart payout. Commands that must be encrypted on encryption-enabled products are highlighted in red. Slave reset Event included in coin mech events. Coin routing example corrected. Corrected Smart payout command examples. Coin acceptance commands added to smart hopper. Note stored event added for smart payout.

Issue 14M – 21/6/09

Change Payout command and event codes to synchronise with Smart hopper codes.
Change DISPENSING event code from D1 to DA due to clash with existing BARCODE_ACK event..
Added EMPTYING and EMPTIED event to Smart Payout and Hopper.
Smart Hopper setup request command code changed from 0x32 to 0x05.
Get Coin Amount command for Smart Hopper changed from 3-byte to 5-byte command.
Set Routing on SMART HOPPER changed from 4 to 6 byte command.
Get Routing 0n SMART HOPPER changed from 3-byte to 5-byte command.

Issue 15 – 23/7/09 Change from Draft to release version.

Issue 16 – 9/9/09

Addition of Get Route command for SMART Payout.
Change note values in SMART payout to be based on penny values not note values so a 5.00 note would be represented in commands and responses by the value 500, not 5 as previously.
Setup request for Bank note validator – response changed to include Real Value Multiplier value (in place of re-teach bytes) to give the full penny values of the payout system.

Issue 17 – 30/11/09

Addition of events in eSSP SMART hopper poll response for connected coin mech jam or coin mech return lever open.
Fraud Attempt event response for Smart Hopper now correctly shows amount dispensed before fraud, not fraud code.

Issue 18 A Draft document for discussion– 16/02/10
Protocol version incremented to 6

BNV command Enable Higher Protocol Events (0x19) is now obsolete.
Multi-country handling added for BNV, Smart Hopper and Smart Payout.
Addition of 'PAYOUT_BY_DENOMINATION' commands for Smart Payout devices.
Addition of 'FLOAT_BY_DENOMINATION' for Smart Payout devices.
Addition of 'PAYOUT_VALUE_TEST' option on payout for Smart Payout devices.
Addition of new PATH_OPEN event in response to poll.
Addition of Multi-currency handling commands/events for Note Validators and Payout devices.
Addition of PAYOUT OUT OF SERVICE event for Smart payout device.
Addition of a Bezel Control command for multi function bezels.
Correction of SET_COIN_INHIBIT command for Smart Hopper – enable/disable polarity was reversed – now correctly displayed as: 0 is disable acceptance, 1 is enable acceptance.
Protocol version 6 now supports full 4-byte channel values on BNV for greater currency value range.
Change to Setup Request Command unit type response codes for Banknote Validators when fitted with Smart Payout and Note Float devices.

Issue 19

- Added Note Float payout device commands and responses.

Issue 20 10/5/2010

- Added new events for Smart Hopper Protocol version 6 – Lid open (0x81), Lid closed (0x82) and Calibration Fail (0x83)
- Removed Coins low (0xD3) and Hopper empty (0xD4 events for Smart Hopper(obsolete in the Smart Hopper).
- Updated Smart Hopper poll response table to version 6 protocol responses.
- Add command SET COMMAND CALIBRATION and RUN COMMAND CALIBRATION for Smart Hoppers
- Expanded and improved SMART HOPPER and SMART PAYOUT poll response descriptions.
- Correction of typo - ASCII code for 'E' was given as 0x44 in some cases – should have been 0x45
- Updated Note Float Spec with version 6 commands and responses

Issue 21 27/5/2010

- Changed INCOMPLETE_PAYOUT response from 0xDD to 0xDC to match Smart Payout response.

Issue 22

- Corrected Smart Hopper Get Coin Amount command value from 2 to 4 bytes

Issue 23

- Added condition for Float by denomination to 0 value
- Added command for Smart hopper to set run time options.

Issue 24

- Added Smart Hopper Coin mech global inhibit command

Issue 25

- SMART Empty command & poll responses for SMART Hopper & SMART Payout
- Channel disable event for all devices except SMART Hopper
- More detailed SMART Payout jam recovery
- Poll response to indicate low number of coins in SMART Hopper
- Updated currency table (Appendix A)
- Corrected poll timeout from 5 to 10 seconds

TABLE OF CONTENTS

INTRODUCTION

This manual describes the operation of the Smiley® Secure Protocol – SSP.

ITL recommend that you study this manual as there are many new features permitting new uses and more secure applications.

If you do not understand any part of this manual please contact the ITL for assistance. In this way we may continue to improve our product. Alternatively visit our web site at www.innovative-technology.co.uk

Enhancements of SSP can be requested by contacting:
support@innovative-technology.co.uk

MAIN HEADQUARTERS	Innovative Technology Ltd Derker Street – Oldham – England - OL1 4EQ Tel: +44 161 626 9999 Fax: +44 161 620 2090 E-mail: support@innovative-technology.co.uk Web site: www.innovative-technology.co.uk

1 GENERAL DESCRIPTION

Smiley® Secure Protocol - SSP is a secure interface specifically designed by ITL® to address the problems experienced by cash handling systems in gaming machines. Problems such as acceptor swapping, reprogramming acceptors and line tapping are all addressed.

The interface uses a master slave model, the host machine is the master and the peripherals (note acceptor, coin acceptor or coin hopper) are the slaves.

Data transfer is over a multi-drop bus using clock asynchronous serial transmission with simple open collector drivers. The integrity of data transfers is ensured through the use of 16 bit CRC checksums on all packets.

Each SSP device of a particular type has a unique serial number; this number is used to validate each device in the direction of credit transfer before transactions can take place. It is recommended that the encryption system be used to prevent fraud through bus monitoring and tapping. This is compulsory for all payout devices.

Commands are currently provided for coin acceptors, note acceptors and coin hoppers. All current features of these devices are supported.

FEATURES:

- Serial control of Note / Coin Validators and Hoppers
- 4 wire (Tx, Rx, +V, Gnd) system
- RS232 (like) - open collector driver
- High Speed 9600 Baud Rate
- 16 bit CRC error checking
- Data Transfer Mode
- Encryption key negotiation
- 128 Bit AES Encrypted Mode

BENEFITS:

- Proven in the field
- Simple and low cost interfacing of transaction peripherals.
- High security control of payout peripherals.
- Defence against surrogate validator fraud.
- Straightforward integration into host machines.
- Remote programming of transaction peripherals
- Open standard for universal use.

To help in the software implementation of the SSP, ITL can provide, C Code, DLL controls and Visual Basic applications on request. Please contact support@innovative-technology.co.uk.

3.0 HARDWARE LAYER

Communication is by character transmission based on standard 8-bit asynchronous data transfer. Only four wires are required TxD, RxD, +V and ground. The transmit line of the host is open collector, the receive line of each peripheral has a 10Kohm pull-up to 5 volts. The transmit output of each slave is open collector, the receive input of the host has a single 3k3 ohm pull-up to 5 volts.

The data format is as follows:

Encoding:	NRZ
Baud Rate:	9600
Duplex:	Full Duplex
Start bits:	1
Data Bits:	8
Parity:	none
Stop bits:	2

Caution:
Power to peripheral devices would normally be via the serial bus however devices that require a high current supply in excess of 1.5 Amps e.g. hoppers would be expected to be supplied via a separate connector.

Recommended Connectors
Two types of connectors are recommended the first is a 15 pin 0.1" pitch header (Molex 22-01-2155), this is primarily for use on bank note acceptors (see table 1).

Pin	Signal
Pin 1	TxD
Pin 5	RxD
Pin 6	Address 0 (Currently not implemented)
Pin 12	GND
Pin 11	+12V
Link Pin 3 to Pin 8.	ENABLE

Table 1 – Bank Note Acceptor Connector Details

The second is a 10-pin 0.1" dual row shrouded header with polarized slot. This is primarily for use with coin acceptors. The pin out is shown below (see table 2).

Pin	Signal	Pin	Signal
1	TxD	2	Reserved
3	RxD	4	Reserved
5	Address 0	6	Address 1
7	+ 12 Volts	8	Ground
9	Address 2	10	Address 4

Table 2 – Coin Acceptor Connector Details

4.0 TRANSPORT LAYER

4.1 PACKET FORMAT

Data and commands are transported between the host and the slave(s) using a packet format as shown below.

STX	SEQ/Slave ID	LENGTH	DATA	CRCL	CRCH

STX:	Single byte indicating the start of a message - 0x7F hex.
SEQ/Slave ID:	Bit 7 is the sequence flag of the packet, bits 6-0 represent the address of the slave the packet is intended for, the highest allowable slave ID is 0x7D
LENGTH:	The length of the data included in the packet - this does not include STX, the CRC or the slave ID.
Slave ID:	Single byte used to identify the address of the slave the packet is intended for.
DATA:	Commands or data to be transferred.
CRCL, CRCH:	Low and high byte of a forward CRC-16 algorithm using the Polynomial ($X^{16} + X^{15} + X^2 +1$) calculated on all bytes, except STX. It is initialised using the seed 0xFFFF. The CRC is calculated before byte stuffing.

4.2 PACKET SEQUENCING

Byte stuffing is used to encode any STX bytes that are included in the data to be transmitted. If 0x7F (STX) appears in the data to be transmitted then it should be replaced by 0x7F, 0x7F. Byte stuffing is done after the CRC is calculated, the CRC its self can be byte stuffed. The maximum length of data is 0xFF bytes. The sequence flag is used to allow the slave to determine whether a packet is a re-transmission due to its last reply being lost. Each time the master sends a new packet to a slave it alternates the sequence flag. If a slave receives a packet with the same sequence flag as the last one, it does not execute the command but simply repeats its last reply. In a reply packet the address and sequence flag match the command packet. This ensures that no other slaves interpret the reply as a command and informs the master that the correct slave replied.

After the master has sent a command to one of the slaves, it will wait for 1 second for a reply. After that, it will assume the slave did not receive the command intact so it will re-transmit it with the same sequence flag. The host should also record the fact that a gap in transmission has occurred and prepare to poll the slave for its serial number identity following the current message. In this way, the replacement of the host's validator by a fraudulent unit can be detected.

The frequency of polling should be selected to minimise the possibility of swapping a validator between polls. If the slave has not received the original transmission, it will see the re-transmission as a new command so it will execute it and reply. If the slave had seen the original command but its reply had been corrupted then the slave will ignore the command but repeat its reply. After twenty retries, the master will assume that the slave has crashed.

A slave has no time-out or retry limit. If it receives a lone sync byte part way through receiving a packet it will discard the packet received so far and treat the next byte as an address byte.

5.0 ENCRYPTION LAYER

5.1 PACKET FORMAT

Encryption is mandatory for all payout devices and optional for pay in devices. Encrypted data and commands are transported between the host and the slave(s) using the transport mechanism described above, the encrypted information is stored in the data field in the format shown below (see figure 1).

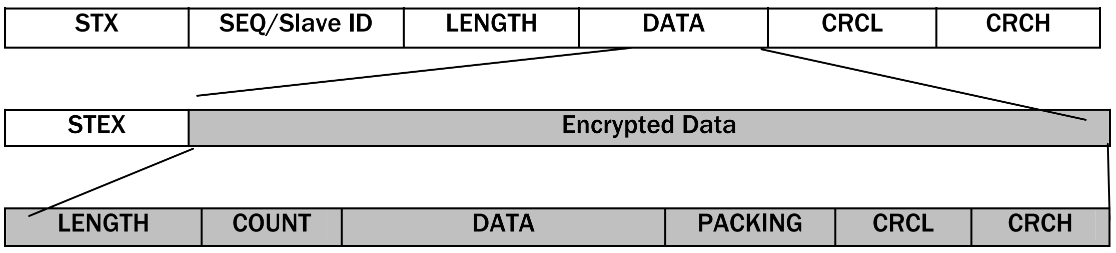


Figure 1 – Encrypted Data Format

STEX:	Single byte indicating the start of an encrypted data block - 0x7E hex.
LENGTH:	The length of the data included in the packet - this does not include STEX, COUNT, the packing or the CRC.
COUNT:	A four byte unsigned integer. This is a sequence count of encrypted packets, it is incremented each time a packet is encrypted and sent, and each time an encrypted packet is received and decrypted.
DATA:	Commands or data to be transferred.
PACKING:	Random data to make the length of the length +count + data + packing + CRCL + CRCH to be a multiple of 16 bytes.
CRCL, CRCH:	Low and high byte of a forward CRC-16 algorithm using the polynomial ($X^{16} + X^{15} + X^2 + 1$) calculated on all bytes, except STEX. It is initialised using the seed 0xFFFF.

After power up and reset the slave will stay disabled and will respond to all commands with the generic response Key_Not_Set, without actioning the command, until the key has been negotiated.

There are two classes of command and response, general commands and commands involved in credit transfer. General commands may be sent with or without using the encryption layer. The slave will reply using the same method, unless the response contains credit information, in this case the reply will always be encrypted. Credit transfer commands, a hopper payout for example, will only be accepted by the slave if received encrypted. Commands that must be encrypted on an encryption-enabled product are highlighted in red throughout this document. The STEX byte is used to determine the packet type. Ideally all communications will be encrypted.

After the data has been decrypted the CRC algorithm is preformed on all bytes including. The result of this calculation will be zero if the data has been decrypted with the correct key. If the result of this calculation is non-zero then the peripheral should assume that the host did not encrypt the data (transmission errors are detected by the transport layer). The slave should go out of service until it is reset.

The packets are sequenced using the sequence count; this is reset to 0 after a power cycle and each time the encryption keys are successfully negotiated. The count is incremented by the host and slave each time they successfully encrypt and transmit a packet and each time a received packet is successfully decrypted.   After a packet is successfully decrypted the COUNT in the packet should be compared with the internal COUNT, if they do not match then the packet is discarded.

5.2 ENCRYPTION KEYS

The encryption key is 128 bits long, however this is divided into two parts. The lower 64 bits are fixed and specified by the machine manufacturer, this allows the manufacturer control which devices are used in their machines. The higher 64 bits are securely negotiated by the slave and host at power up, this ensures each machine and each session are using different keys. The key is negotiated by the Diffie-Hellman key exchange method. See: http://en.wikipedia.org/wiki/Diffie-Hellman. The exchange method is summarised in the table below. C code for the exchange algorithm is available from ITL.

	Host	Slave
1	Generate prime number GENERATOR	
2	Use command 'Set Generator' to send to	
3	slave	Check GENERATOR is prime and store
4		
5	Generate prime number MODULUS	
6	Use command 'Set Modulus' to send to	Check MODULUS is prime and store
7	slave	
8		
	Generate Random Number HOST_RND	
9	Calculate HostInterKey: =	
	GENERATOR ^ HOST_RND mod	
10	MODULUS	Generate Random Number SLAVE_RND
11	Use command 'Request Key Exchange to	Calculate SlaveInterKey: =
	send to slave.	GENERATOR ^ SLAVE_RND mod
12		MODULUS
		Send to host as reply to 'Request Key
		Exchange'
13		
		Calculate Key: =
		HostInterKey ^ SLAVE_RND mod
	Calculate Key: =	MODULUS
	SlaveInterKey ^ HOST_RND mod	
	MODULUS	

Note: '^' represents 'to the power of'

Action	Command code (HEX)
Set Generator	0x4A, Generator
Set Modulus	0x4B, Modulus
Request Key Exchange	0x4C, HostInterKey

Table 3 - Encryption Control Commands

Set Generator: The Nine byte command, the first byte is the command –0x4A. The next eight bytes are a 64 bit number representing the Generator this must be a 64bit prime number. The slave will reply with OK or 'Parameter out of range' if the number is not prime.

Set Modulus: The Nine byte command, the first byte is the command –0x4B. The next eight bytes are a 64 bit number representing the modulus this must be a 64bit prime number. The slave will reply with OK or 'Parameter out of range' if the number is not prime.

Request Key Exchange: Nine byte command, the first byte is the command –0x4C. The next eight bytes are a 64 bit number representing the Host intermediate key. If the Generator and Modulus have been set the slave will calculate then reply with the generic response and eight data bytes representing the slave intermediate key. The host and slave will then calculate the key. If Generator and Modulus are not set then the slave will reply FAIL.

5.3 ENCRYPTION ALGORITHM

The encryption algorithm used is AES with a 128-bit key; this provides a very high level of security. Data is encrypted in blocks of 16 bytes any unused bytes in a block should be packed with random bytes. AES is used in electronic codebook mode (ECB). Please contact ITL for an implementation of key exchange and AES encryption, this is provided as C source code. See: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

The encryption functions included in issue 14 and earlier of this document are redundant.

6.0 CONTROL LAYER

6.1 INTRODUCTION

The slave can only respond to requests from the master with an address byte that matches the slaves address, at no time will the slave transmit any data that is not requested by the host. Any data that is received with an address that does not match the slave's address will be discarded.

The master will poll each slave at least every 10 seconds. The slave will deem the host to be inactive, if the time between polls is greater than 10 seconds. If the slave does not receive a poll within 10 seconds it should change to its disabled state. The minimum time between polls is specified for individual peripherals. Only one command can be sent in any one poll sequence.

6.2 ADDRESSING

The address of a peripheral consists of two parts, the fixed part that determines the type of device and the variable part. The variable part is used if there is a number of the same type of peripheral in the same machine, for example hoppers (see table 4).

The variable part of the address can be set in one of two ways. Firstly it can be programmed to a fixed number using a PC tool, or the peripheral can be programmed to take the rest of the address from external pins on the interface connector (Currently not implemented).

Slave ID (Hex)	Peripheral
0x00	Note validator 0
0x01	Note validator 1
0x02	Coin Validator 0
0x03	Coin Validator 1
0x04	Card Reader 0
0x05	Card Reader 1
0x07	Audit Device
0x08	Handheld Audit Collection Device
0x09 – 0x0F	Reserved
0x10 – 0x1F	Coin Hoppers 0 – 15
0x20 – 0x2F	Note Dispensers 0 – 15
0x30 – 0x3F	Card Dispensers 0 – 15
0x40 – 0x4F	Ticket Dispensers 0 – 15
0x50 – 0x5F	Extra Note Validators
0x60 – 0x7E	Unallocated

Table 4 – Peripheral Addressing

6.3 PERIPHERAL VALIDATION.

To ensure that credit transfers are only received from or sent to genuine devices, the device receiving the credit must first request the serial number from the sending device and only accept if the serial number matches a pre-programmed number.

The serial number should be requested after each reset and also after each break in communications. For example a host machine should request a coin acceptors serial number at reset or if a poll sequence is unanswered, before enabling the device. Also, a coin hopper should not process any dispense commands until the host machine has sent its serial number.

6.4 GENERIC COMMANDS AND RESPONSES.

Generic commands are a set of commands that every peripheral must understand and act on (see table 5).

6.4.1 GENERIC COMMANDS

Action	Command code (HEX)
Reset	0x01
Host Protocol Version	0x06
Poll	0x07
Get Serial Number	0x0C
Synchronisation command	0x11
Disable	0x09
Enable	0x0A
Program Firmware / currency	0x0B, Programming Type
Manufactures Extension	0x30, Command, Data

Table 5 – Generic Commands

Reset: Single byte command, causes the slave to reset.
Host Protocol Version: Dual byte command, the first byte is the command; the second byte is the version of the protocol that is implemented on the host. So for example to enable events on BNV to protocol version 6, send 06, 06. The device will respond with OK if the device supports version 6, or FAIL (0xF8) if it does not.
This command should be sent as part of the start up sequence in order that any new start-up events are not missed.
Poll: Single byte command, no action taken except to report latest events.
Get Serial Number: Single byte command, used to request the slave serial number.
Returns 4-byte long integer.

Most significant byte first e.g.
Serial number = 01873452 = 0x1C962C
So response data would be 0x00 0x1C 0x96 0x2C

Sync: Single byte command, which will reset the validator to expect the next sequence ID to be 0.
Disable: Single byte command, the peripheral will switch to its disabled state, it will not execute any more commands or perform any actions until enabled, any poll commands will report disabled.
Enable: Single byte command, the peripheral will return to service.
Program Firmware / currency: See section 6.4.3 – Remote Programming.
Manufactures Extension: This command allows the manufacturer of a peripheral to send commands specific to their unit. The intention is that the manufacturer only uses the extension command internally; it should not when operating in a host machine. The specific command and any data for that command should follow the Extension command.

6.4.2 GENERIC RESPONSES

Generic Response	Response code
OK	0xF0
Command not known	0xF2
Wrong number of parameters	0xF3
Parameter out of range	0xF4
Command cannot be processed	0xF5
Software Error	0xF6
FAIL	0xF8
Key Not Set	0xFA

Table 6 - Generic Responses

OK: Returned when a command from the host is understood and has been, or is in the process of, being executed.
Command Not Known: Returned when an invalid command is received by a peripheral.
Wrong Number Of Parameters: A command was received by a peripheral, but an incorrect number of parameters were received.
Parameter Out Of Range: One of the parameters sent with a command is out of range. E.g. trying to change the route map for channel 34 on a coin acceptor.
Command Cannot Be Processed: A command sent could not be processed at that time. E.g. sending a dispense command before the last dispense operation has completed.
Software Error: Reported for errors in the execution of software e.g. Divide by zero. This may also be reported if there is a problem resulting from a failed remote firmware upgrade, in this case the firmware upgrade should be redone.
Key Not Set: The slave is in encrypted communication mode but the encryption keys have not been negotiated.

6.4.3 REMOTE PROGRAMMING.

Code	Description
0x0B, Type	Start Programming, type (00 – firmware, 01 - currency)
0x16	Programming Status.

Table 7 - Remote Programming Code Summary

Using the command 0x0B followed by a parameter that indicates the type of programming required performs remote programming (see table 7). Send 0x00 for firmware programming and 0x01 for currency data programming.
The peripheral will respond with a generic reply. If the reply is OK, the host should send the first block of the data file (the file header). The header has the format shown below (see table 8). The block size depends on the peripheral used but must be a minimum of 10 bytes to contain the header data.

When the block size for a peripheral is greater than the header length (11 bytes) then the header is padded out with 0's to the length of a block. The maximum length of a block is 236 bytes.

File offset	Description	Size
0	Number of blocks to send (low byte, high byte), including header block	2 bytes
2	Manufacture code (of file) e.g. 'ITL'	3 bytes
5	File type – 0x00 firmware, 0x01 currency	1 byte
6	Unit subtype	1 byte
7	Unit version	1 byte
8	Block length (B_L)	1 byte
9	Checksum (CRC of data section of file) CRC low byte	1 byte
A	Checksum (CRC of data section of file) CRC high byte	1 byte
B	Padded 0's to block size	B_L-11 bytes

Table 8 - Remote Programming / Header and Block Size

The peripheral will then respond with OK or HEADER_FAIL depending on the acceptability of this file (see table 9).

Response	Code
OK	0xF0
HEADER_FAIL	0xF9

Table 9 – Peripheral Response

The host will then send the required number of data blocks. The peripheral will respond with a generic response when each packet has been processed (see table 10).

If the host receives any response other than an OK then that packet is retried three times before aborting the programming (the peripheral should then be reset).

After the last data packet has been sent and a response received, the host will send a programming status command 0x16. The peripheral will respond with one of the following codes:

Response	Code
OK	0xF0
Checksum Error	0xF7
FAIL	0xF8

Table 10 - Peripheral Response Codes

After a successful programming cycle, the peripheral should be reset. If the programming cycle does not complete successfully, then the peripheral should be disabled until it can be programmed successfully.

In the case of an unsuccessful firmware programming cycle, the new firmware will either be discarded or partly programmed. If the firmware has been partly programmed, then the peripheral will respond to all Polls with the generic response 'Software Error'.

The peripheral will not allow the host to enable it until it receives a complete and valid firmware file.

6.4.5 SIMPLIFIED REMOTE PROGRAMMING FLOW CHART.

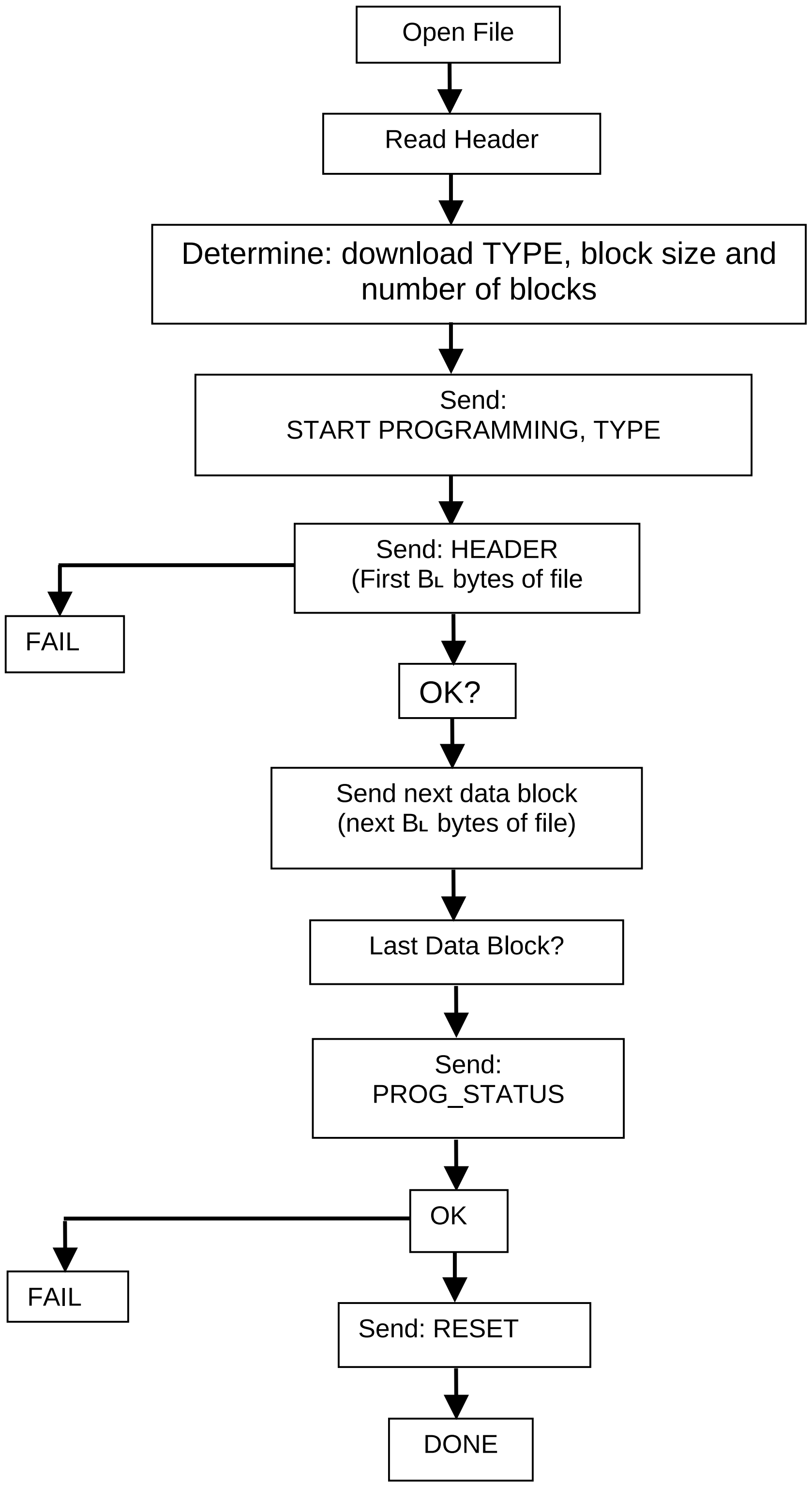

Open File
Read Header
Determine: download TYPE, block size and number of blocks
Send: START PROGRAMMING, TYPE
Send: HEADER (First BL bytes of file
FAIL
OK?
Send next data block (next BL bytes of file)
Last Data Block?
Send: PROG_STATUS
OK
FAIL
Send: RESET
DONE

Figure 2 - Programming Flow Chart

6.5 BANKNOTE VALIDATOR

6.5.1 BNV OPERATION.

When the validator has recognised a note, it will not start to stack it until it receives the next valid poll command after the read n (n<>0) has been sent. The note will be rejected if the host responds with a REJECT.

6.5.2 BNV COMMANDS

Action	Command code (HEX)
Set inhibits	0x02
Display On	0x03
Display Off	0x04
Set-up Request	0x05
Reject	0x08
Unit data	0x0D
Channel Value data	0x0E
Channel Security data	0x0F
Channel Re-teach data	0x10
Last Reject Code	0x17
Hold	0x18
Enable Protocol Version Events	0x19 (made obsolete in protocol version 6)
Get Bar Code Reader Configuration	0x23
Set Bar Code Reader Configuration	0x24
Get Bar Code Inhibit	0x25
Set Bar Code Inhibit	0x26
Get Bar Code Data	0x27

Table 13 – Bank Note Validator Commands

Set Inhibits: Variable length command, used to control which channels are enabled. The command byte is followed by 2 data bytes, these bytes are combined to create the INHIBIT_REGISTER, each bit represents the state of a channel (LSB= channel 1, 1=enabled, 0=disabled). At power up all channels are inhibited and the validator is disabled.
Display On: Single Byte command, turns on the display illumination bulb.
Display Off: Single Byte command, turns off the display illumination bulb.
Reject: Single byte command causing the validator to reject the current note.
Set-up Request: Single byte command, used to request information about a slave. Slave will return the following data: Unit Type, Firmware version, Country Code, Value multiplier, Number of channels, (if number of channels is 0 then 0 is returned and next two parameters are not returned) Value per channel, security of channel, Reteach count, Version of Protocol. If protocol version is greater or equal to 5, then additional bytes here will give the country code for each channel (see table 14) and expanded 4-byte channel value bytes.
When the host and slave support protocol version >= 6, the expanded channel values can be used to give a greater range. In this case, the Value Multiplier bytes will be 0 and the channel values will given a 4-byte little endian number at the end of the setup data response. For example a channel value of 500,000 would be 20 A1 07 00.

Data	Size/type	Table Offset	Notes
Unit Type	1 byte	0	0x00 Note Validator 0x06 Smart Payout fitted 0x07 Note Float device fitted
Firmware Version	4 bytes	1	XX.XX (can include space)
Country Code	3 bytes	5	See Country Code Table
Value Multiplier	3 bytes	8	24 bit value (if 0 then use expanded values for channel value – Protocol V6 only)
Number of channels	1 byte	11	Highest used channel (n)
Channel Value	n bytes	12	bytes 1 – 15 values (0 if expanded values used)
Security of Channel	n bytes	12 + n	bytes 1 – 15 security (0 if expanded values used)
Real value multiplier	3 byte	12 + (n*2)	The value by which the channel values can be multiplied to show the true value.
Protocol version	1 byte	15 + (n*2)	Current protocol version of this BNV
Channel Country Code **(protocol version >= 6 only)**	3 x n bytes	16 + (n*2)	3 byte ASCII country code for each channel
Expanded channel values **(protocol version >= 6 only)**	4 x n bytes	16 + (n*5)	Full channel values (4 byte integers)

Table 14 - Response to Set-up request

Unit Data Request: Single byte command which returns, Unit type (1 Byte integer), Firmware Version (4 bytes ASCII string), Country Code (3 Bytes ASCII string), Value Multiplier (3 bytes integer), Protocol Version (1 Byte, integer)
Channel Value Request: Single byte command which returns a number of channels byte (the highest channel used) and then 1 to n bytes which give the value of each channel up to the highest one, a zero indicates that the channel is not implemented.
e.g. A validator has notes in Channels 1,2,4,6,7 so this command would return 07,01,02,00,04,00,06,07.
The actual value of a note is calculated by multiplying the value multiplier by channel value.
If the number of channels is 0 then only one 0 will be returned.
When the host and slave support protocol version >= 6, the channel values response will be given as:
Highest Channel, Value Per Channel (0 for expanded values),3 Byte ASCI country code for each channel, 4- byte Full channel Value for each channel.

Channel Security Data: Single byte command which returns a number of channels byte (the highest channel used) and then 1 to n bytes which give the security of each channel up to the highest one, a zero indicates that the channel is not implemented.
(1 = low, 2 = std, 3 = high, 4 = inhibited).
E.g. A validator has notes in Channels 1,2,4,6,7 channel 1 is low security, channel 6 is high security, all the rest are standard security.
The return bytes would be
07,01,02,00,02,00,02,03
If the number of channels is 0 then only one 0 will be returned.

Real Value Multiplier: Use this value as a multiplier for channel values to find the payout values to use if this validator is used with a SMART payout system.

Last Reject Code: Single byte command, which will return a single byte that indicates the reason for the last reject. The codes are shown below (see table 15). Specifics of note validation are not shown to protect integrity of manufacturers security.

Code	Reject Reason
0x00	Note Accepted
0x01	Note length incorrect
0x02	Reject reason 2
0x03	Reject reason 3
0x04	Reject reason 4
0x05	Reject reason 5
0x06	Channel Inhibited
0x07	Second Note Inserted
0x08	Reject reason 8
0x09	Note recognised in more than one channel
0x0A	Reject reason 10
0x0B	Note too long
0x0C	Reject reason 12
0x0D	Mechanism Slow / Stalled
0x0E	Strimming Attempt
0x0F	Fraud Channel Reject
0x10	No Notes Inserted
0x11	Peak Detect Fail
0x12	Twisted note detected
0x13	Escrow time-out
0x14	Bar code scan fail
0x15	Rear sensor 2 Fail
0x16	Slot Fail 1
0x17	Slot Fail 2
0x18	Lens Over Sample
0x19	Width Detect Fail
0x1A	Short Note Detected

Table 15 – Reject Code Reasons

Hold: This command may be sent to BNV when Note Read has changed from 0 to >0 (valid note seen) if the user does not wish to accept or reject the note with the next command.
This command will also reset the 10-second time-out period after which a note held would be rejected automatically, so it should be sent before this time-out if an escrow function is required.
Enable higher protocol version events: Single byte command to enable events implemented in protocol version >=3. Send this command directly as part of the start-up routine before any POLLS are sent to ensure any new events are seen. If Command is not known (0xF2) is returned, this feature is not implemented in the firmware. Otherwise a two-byte response will return OK, and then the current protocol version of the validator being addressed. This command is now obsolete in protocol version 6. Devices will respond to this command by enabling events up to version 4. For any new events, please use the Generic command Host Protocol Version 0x06.

Get Bar Code Reader Configuration: Single byte command, returns generic response + configuration data for Bar code reader.

Data byte	
0	Bar code hardware status (0x00 = none, 0x01 = Top reader fitted, 0x02 = Bottom reader fitted, 0x03 = both fitted)
1	Readers enabled (0x00 = none, 0x01 = top, 0x02 = bottom, 0x03 = both)
2	Bar code format (0x01 = Interleaved 2 of 5)
3	Number of Characters (Min 6 max 24)

Set Bar Code Reader Configuration: 0x23 + 3 byte command data:

Command byte	
0	0x00 Enable none, 0x01 enable top, 0x02 = enable bottom, 0x03 = enable both
1	Bar code format (0x01 = Interleaved 2 of 5)
2	Number of characters (Min 6 Max 24)

Get Bar Code Inhibit: Single byte command to return the current bar code/currency inhibit status.
Data byte return is a bit register formatted as:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	1	1	1	1	Bar	Currency

Bit 0 : Currency enable 0 = enable, 1 = disable
Bit 1: Bar code ticket enable 0 = enable, 1 = disable
Example 0xFE is barcode inhibited, currency enabled. (Default state)

Set Bar Code Inhibit: Command byte plus 1 data byte to set the Bar code/ currencies inhibit status. Note that the reset default state is 0xFE – currency enabled, Bar code disabled.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	1	1	1	1	Bar	Currency

Bit 0 : Currency enable 0 = enable, 1 = disable
Bit 1: Bar code ticket enable 0 = enable, 1 = disable

Get Bar Code Data: Single byte command to obtain last valid bar code ticket data, send in response to a Bar Code Ticket Validated event. This command will return a variable length data steam, a generic response (OK) followed by a status byte, a bar code data length byte, then a stream of bytes of the ticket data in ASCII.

Status byte: 0x00 – no valid data, 0x01 ticket in escrow, 0x02 ticket stacked, 0x03 ticket rejected
For example:
Command data response 0xF0,0x01,0x06,0x31,0x32,0x33,0x34,0x35,0x36 is:
Ticket in escrow with data length: 6 Ticket data: '123456'

6.5.3 BNV RESPONSE TO POLLS

In response to any command from the master, the slave will respond with a generic response and some data (see table 16). If the command is a poll then a message containing a list of events that have occurred since the last poll, each event can only occur once in each response packet.

Event/ State	Event Code	Minimum Protocol Version
Slave Reset	0xF1	
Read, n	0xEF, Channel No	
Credit, n	0xEE, Channel No	
Rejecting	0xED	
Rejected	0xEC	
Stacking	0xCC	
Stacked	0xEB	
Safe Jam	0xEA	
Unsafe Jam	0xE9	
Disabled	0xE8	
Fraud Attempt, n	0xE6, Channel No	
Stacker Full	0xE7	
Note cleared from front at reset	0xE1, Channel No	4
Note cleared into cash box at reset	0xE2, Channel No	4
Cash Box Removed	0xE3	4
Cash Box Replaced	0xE4	4
Bar Code Ticket Validated	0xE5	4
Bar Code Ticket Acknowledge	0xD1	4
Note path open	0xE0	6
Channel Disable	0xB5	7

Table 16 – BNV Response Codes

Slave Reset: Returned when a peripheral has just powered up or when the host has sent a reset command.
Read: The slave is reading a note, the second byte indicates which channel the note belongs to, if the channel is currently unknown then zero is returned.
Credit: The slave has accepted currency on the channel indicated; the currency is now past the point where the customer can recover the currency. The credit event is only sent once (except where communication fails). The second byte indicates the channel of the credit.
Rejecting: The validator is currently rejecting a note.
Rejected: The slave has rejected the currency that was entered.
Stacking: The slave is moving the currency to a secure location.
Stacked: The stacking unit has completed its cycle.
Safe Jam: The slave has jammed and cannot return to service, the user cannot retrieve a note and a credit could have been given.
Unsafe Jam: The slave is jammed and cannot return to service, the credit has not been given and the user may be able to retrieve the note.
Disabled: The slave has been disabled, either by disable command, disabling all channels or the 10 second poll time out has expired.
Note cleared from front at reset: The validator has detected that a not was in the path at start up and has attempted to clear it from the front of the BNV. If its channel was known, the channel No will be greater than 0. This event is only reported if the Host Protocol Version (0x06) command was sent in the start-up sequence to enable events greater or equal to 4.
Note cleared into cash box at reset: The validator has detected that a note was in the path at start up and has attempted to clear it into the cash box. If its channel was known, the channel number will be greater than 0. This event is only reported if the Host Protocol Version (0x06) command was sent in the start-up sequence to enable events greater or equal to 4.
Fraud Attempt: The validator has detected an attempt to fish notes out of the Stacker.
Cash Box Removed: This event is only reported if the Host Protocol Version (0x06) command was sent in the start-up sequence to enable events greater or equal to 4.
Cash Box Replaced: This event is only reported if the Host Protocol Version (0x06) command was sent in the start-up sequence to enable events greater or equal to 4.
Bar Code Ticket Validated: The validator has detected and validated a TITO ticket and it is held in escrow. The ticket details can be obtained by sending GET_BAR_CODE_DATA command. The ticket will be stacked on the next poll command or can be rejected with REJECT command.
Bar Code Ticket Acknowledge: The bar code ticket has reached its safe stack point (equivalent to note credit event).
Note path open: This event is only reported if the Host Protocol Version (0x06) command was sent in the start-up sequence to enable events >= 6. It will be reported if the BNV detects that its note path is not completely closed thus disabling the BVN for operation.
Channel Disable: Indicates all note channels have been inhibited and as such, the unit is disabled. Only reported if using protocol version 7 and above.

6.6 COIN ACCEPTOR

6.6.1 COIN ACCEPTOR OPERATION

The coin acceptor will provide generic response polls to commands from the master, the acceptor will respond with a response and some data (see table 17).

6.6.2 COIN ACCEPTOR COMMANDS

Action	Command code (HEX)
Set inhibits	0x02
Set-up Request	0x05
Unit data	0x0D
Channel Value data	0x0E
Channel Security data	0x0F
Channel Re-teach data	0x10
Last Reject Code	0x17
Update Coin Route	0x12

Table 17 – Coin Acceptor Commands

Set Inhibits: Variable length command, used to control which channels are enabled. The command byte is followed by n data bytes, these bytes are combined to create the INHIBIT_REGISTER, each bit represents the state of a channel (LSB= channel 1, 1=enabled, 0=disabled). At power up all channels are inhibited and the validator is disabled.

Set-up Request: Single byte command, used to request information about a slave. Slave will return the following data (see table 18): Unit Type, Firmware version, Country Code, Value multiplier, Number of channels, (if number of channels is 0 then 0 is returned and next two parameters are not returned) Value per channel, security of channel, Re-teach count, Version of Protocol.

Data	Size/type	Notes
Unit Type	1 byte, integer	0x01 Coin Validator
Firmware Version	4 bytes, string	XX.XX (can include space)
Country Code	3 bytes, string	See Country Code Table
Value Multiplier	3 bytes, integer	24 bit value
Number of channels	1 byte, integer	Highest used channel
Channel Value	15 byte, integer	bytes 1 - n values
Security of Channel	15 byte, integer	bytes 1- n security
Reteach count	3 byte, integer	Byte 1 - reteach count. Byte 2,3 flag register indicating which channels have been modified. All set to zero at factory.
Protocol version	1 byte, integer	

Table 18 – Response to Set-up request

Unit Data Request: Single byte command which returns, Unit type (1 Byte integer), Firmware Version (4 bytes ASCII string), Country Code (3 Bytes ASCII string), Value Multiplier (3 bytes integer), Protocol Version (1 Byte, integer)
Channel Value Request: Single byte command which returns a number of channels byte (the highest channel used) and then 1 to n bytes which give the value of each channel up to the highest one, a zero indicates that the channel is not implemented.
E.g. A validator has coins in Channels 1,2,4,6,7 so this command would return 07,01,02,00,04,00,06,07. (The values are just examples and would depend on the currency of the unit). The actual value of a coin is calculated by multiplying the value multiplier by channel value.

If the number of channels is 0 then only one 0 will be returned.
Channel Security Data: Single byte command which returns a number of channels byte (the highest channel used) and then 1 to n bytes which give the security of each channel up to the highest one, a zero indicates that the channel is not implemented.
(1 = low, 2 = std, 3 = high, 4 = inhibited).
E.g. A validator has coins in Channels 1,2,4,6,7 channel 1 is low security, channel 6 is high security , all the rest are standard security.
The return bytes would be
07,01,02,00,02,00,02,03
If the number of channels is 0 then only one 0 will be returned.
Channel Reteach Data: Single byte command, which returns 3 bytes.
First byte - the number of times the unit has been manually taught (1 for each face).
Second byte - Channels 1 to 8 flag register bit 0 = channel 1 to bit 7 = channel 8 if set shows that the indicated channel has been altered.
Third byte is as second but the channels shown are bit 1 = channel 9 to bit 6 = channel 15.
Last Reject Code: Single byte command, which will return a single byte that indicates the reason for the last reject. The codes are shown below (see table 19).

Code	Reject Reason
0x00	Coin Accepted
0x01	Reject error 1
0x02	Reject error 2
0x03	Reject error 3
0x04	Reject error 4
0x05	Channel Inhibited (software)
0x06	Channel Inhibited (SSP)
0x07	Closely following coin
0x08	Reject error 8
0x09	Match in multiple windows
0x0A	Reject error 10
0x0B	Reject error 11
0x0C	Reject error 12
0x0D	Reject error 13
0x0E	Strim Attempt
0x0F	Fraud Channel Reject
0x10 – 0xFF	Reserved

Table 19 - Reject Reason Codes

Update Coin Route: This command consists of three bytes; the actual command, the channel to be updated and the route information. The route information is packed into one byte, each bit represents a route, if a bit is set then the route is allowed for that channel.
Example: 0x12, 0x02, 0x04 – Update route for channel 2 to 00000100 (route 3).

6.6.3 COIN ACCEPTOR RESPONSES TO POLLS

In response to any command from the master the acceptor will respond with a generic response and some data (see table 20). If the command is a poll then a message containing a list of events that have occurred since the last poll, each event can only occur once in each response packet.

Event / State	Event Code
Slave Reset	0xF1
Credit, n	0xEE, CHANNEL Nº
Rejected	0xEC
Coin Routed	0xE5, route Nº
Safe Jam	0xEA
Unsafe Jam	0xE9
Disabled	0xE8
Fraud Attempt	0xE6
Channel Disable	0xB5

Table 20 – Coin Acceptor Response Codes

Credit: The slave has accepted currency on the channel indicated; the currency is now past the point where the customer can recover the currency. The credit event is only sent once (except where communication fails). The second byte indicates the channel of the credit.
Note: It is possible that there may be more than one credit event in each packet.
Rejected: The slave has rejected the currency that was entered.
Coin Routed: The acceptor has routed the last coin accepted.
Safe Jam: The slave has jammed and cannot return to service, the user cannot retrieve a coin and a credit has been given.
Unsafe Jam: The slave is jammed and cannot return to service, the credit has not been given and the user may be able to retrieve the coin.
Disabled: The slave has been disabled, either by disabling all channels or the 10 second poll time out has expired.
Fraud Attempt: The validator has detected an attempt to fish coins out of the unit.
Channel Disable: Indicates all note channels have been inhibited and as such, the unit is disabled. Only reported if using protocol version 7 and above.

6.7 SINGLE COIN HOPPER

6.7.1 SINGLE COIN HOPPER OPERATION

All transactions with a hopper should be encrypted to prevent dispense commands being recorded and replayed by an external device.

To enable the dispense command of the hopper the host must send it a serial number, which matches the one stored in the hopper. This has two implications for the hopper firstly it must have some non-volatile memory to store the correct serial number in. Secondly, it must have a means of switching mode so that the next serial number received is stored as its reference number.

This allows the hopper to be installed in a host machine without manually programming the reference serial number. As the hopper is in a secure location in the host, this could be done by pressing a recessed button in the hopper.

6.7.2 SINGLE COIN HOPPER COMMANDS

Action	Command code (HEX)
Dispense	0x13, Nº of coins
Host Serial Number Request	0x14, Serial Nº
Set-up Request	0x15

Table 21 – Coin Hopper Commands

Dispense: Two-byte command, the first byte is the command it's self and the second is the number of coins to dispense.
Host serial number request: This allows the host machine to send its serial number to the hopper. After a reset or break in communications the hopper will not process any other commands until it has received a serial number equal to the one it has in its memory. If more than ten invalid serial numbers are received, the hopper will go out of service until its set-up input is activated.
Set-up Request: Single byte command, used to request information about a slave (see table 22). Slave will return the following data: Unit Type, Firmware version, Country Code, Coin Type, Maximum Capacity, Version of Protocol.

Data	Size / type	Notes
Unit Type	1 byte, integer	0x02 Coin Hopper
Firmware Version	4 bytes, string	XX.XX (can include space)
Country Code	3 bytes, string	See Country Code Table
Coin Type	1 byte, integer	
Maximum Capacity	2 Bytes, integer	
Low coin value	1 Byte, integer	Number of coins left at low coin event.
Protocol version	1 byte, integer	

Table 22 - Response to Set-up request

6.7.3 SINGLE COIN HOPPER RESPONSES TO POLLS

Event/ State	Event Code
Dispensing	0xD1, No of coins dispensed
Dispensed	0xD2, No of coins
Coins Low	0xD3
Empty	0xD4
Jammed	0xD5
Fraud Attempt	0xE6, Fraud Code

Table 23 – Coin Hopper Responses to Polls

Dispensing: Two-byte response the second byte is the number of coins that have been dispensed at the point when the poll was received.
Dispensed: Two-byte response that indicates when the hopper has finished a dispense operation, either because the required number of coins have been dispensed or the hopper is out of coins or jammed. The second byte is the number of coins that were successfully dispensed.
Coins Low: This is reported when the hopper has become low on coins, the hopper will report this event until it is empty or refilled.
Coins Empty: Single byte response indicating that the hopper is empty of coins; the hopper will report this state until it is filled it will also become disabled.
Jammed: Single byte response that indicates that the hopper is jammed; this is reported until it is un-jammed or reset. It will also become disabled.
Fraud Attempt: This will be reported if an attempt has been made to remove coins from the hopper.

6.8 BASIC CARD READER.

No longer supported

6.9 SMART HOPPER

6.9.1 SMART HOPPER OPERATION

All transactions with a hopper should be encrypted to prevent dispense commands being recorded and replayed by an external device.

6.9.2 SMART HOPPER COMMANDS

Action	Command code (HEX)	Data 1	Data 2	Data 3	Data 4
GET DEVICE SETUP	0x05	-	-	-	-
SET ROUTING	0x3B	Route, 1 byte	Value, 2 bytes	*Country code, 3 Bytes*	-
GET ROUTING	0x3C	Value, 4 bytes	*Country code, 3bytes*	-	-
PAYOUT AMOUNT	0x33	Value, 4 bytes	*Country code, 3 Bytes*	*Option, 1 byte*	-
GET COIN AMOUNT	0x35	Value, 4 Bytes	*Country code, 3 bytes*	-	-
SET COIN AMOUNT	0x34	Amount, 2 bytes	Value, 2 bytes	*Country code, 3 Bytes*	
HALT PAYOUT	0x38	-	-	-	-
FLOAT AMOUNT	0x3D	Min payout, 2 bytes	Value, 4 bytes	*Country Code, 3 Bytes*	*Option, 1 byte*
GET MINIMUM PAYOUT	0x3E	*Country Code, 3 bytes*	-	-	-
SET COIN MECH INHIBTS	0x40	State, 1 byte	Value,, 2 bytes	*Country Code, 3 Bytes*	
PAYOUT BY DENOMINATION	0x46	See detail			
FLOAT BY DENOMINATION	0x44	See detail			
SET COMMAND CALIBRATION	0x47	Mode, 1 Byte			
RUN COMMAND CALIBRATION	0x48				
EMPTY ALL	0x3F				
SET OPTIONS	0x50	Option 0	Option 1		
GET OPTIONS	0x51				
COIN MECH GLOBAL INHIBIT	0x49	Mode, 1 byte	-	-	-
SMART EMPTY	0x52				
CASHBOX PAYOUT OPERATION DATA	0x53	See detail			

Table 21 – Smart Hopper Commands

Important note about version 6 protocol:
If the Payout unit supports protocol version 6 (Use the SET_UP_REQUEST 0x05 command to determine version) and the host has enabled protocol version 6 commands and events using the generic command HOST_PROTOCOL_VERSION 0x06, the host can send the additional parameters for country code and option to use the multi-currency and payout test amount facilities in this updated protocol.

Notice about sending coin values:
All coin values to and from the payout are formatted as 4 byte integer values. For example to send a coin value of 5.20, we get the full value (x100) = 520, convert this to hex = 0208. We then send this as a 4 byte little endian order array – 08 02 00 00

GET DEVICE SETUP 0x05:
Single byte command, used to request information about a slave (see table 22). Slave will return the following data: Unit Type, Firmware version, Country Code, Version of Protocol, number of different coin values and the value of each coin.

Data	Size	Table offset	Notes
Unit Type	1 byte	0	0x03 Smart Hopper
Firmware Version	4 byte	1	XX.XX (can include space)
Main Country Code	3 bytes	5	See Country Code Table. E.g. GBP
Protocol version	1 byte	8	
Number of Coin Values (n)	1 byte	9	
Coin Values	2 * n bytes	10	e.g. 0x01, 0x00 = 0.01
Country codes for values (protocol version 6 only)	3 * n bytes	10 + (2*n)	e.g. 'EUR'

Table 22 - Response to Set-up request

SET ROUTING 0x3B:
A command to setup the desired route of a coin entered into the hopper unit. Note that protocol 6 version commands have been expanded to use a 4-byte coin value.

Route	code (HEX)
Coins recycled and used for payouts	0x00
Coins routed to cashbox	0x01

Example:
To send 0.10 EUR coins to the Hopper cashbox route:
Version 5 protocol:
3B 01 0A 00
Version 6 protocol:
3B 01 0A 00 00 00 45 55 52

GET ROUTING 0x3C:

A command to determine the route setting of a coin value. Note that protocol 6 version commands have been expanded to use a 4-byte coin value.

Example:
To read the route of a 1.00 EUR coin:
Version 5 protocol:
    3C 64 00
Version 6 protocol:
    3C 64 00 00 00 45 55 52

The command will return the route as a single byte value as below:

Route	code (HEX)
Coin recycled and used for payouts	0x00
Coin routed to cashbox	0x01

PAYOUT AMOUNT 0x33:

A command to set the monetary value to be paid by the payout unit. Using protocol version 6, the host also sends a pre-test option byte (TEST_PAYOUT_AMOUT 0x19, PAYOUT_AMOUNT 0x58), which will determine if the command amount is tested or paid out. This is useful for multi-payout systems so that the ability to pay a split down amount can be tested before committing to actual payout.

Example:
To payout 25.20 EUR:
Version 5 protocol to payout
    33 D8 09 00 00
Version 6 Protocol to test payout ability
    33 D8 09 00 00 45 55 52 19
Version 6 Protocol to payout
    33 D8 09 00 00 45 55 52 58

If the payout is possible the payout will reply with generic response OK. If the payout is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Payout	0x01
Cant pay exact amount	0x02
Smart Payout Busy	0x03
Smart Payout Disabled	0x04

GET COIN AMOUNT 0x35:
A command to return the amount of coins stored of a given denomination in the payout unit.
Example – To get the number of 0.20 EUR coins stored
Version 5 protocol:

```
35 14 00 00 00
```

Version 6 protocol:

```
35 14 00 00 00 45 55 52
```

SET COIN AMOUNT 0x34:
A command to set the level of coins of a denomination stored in the hopper. The command is formatted with the command byte first, amount of coins to add as a 2-byte little endian, the value of coin as 2-byte little endian and (if using protocol version 6) the country code of the coin as 3 byte ASCII.
The level of coins for a denomination can be set to zero by sending a zero level for that value.
Note that protocol 6 version commands have been expanded to use a 4-byte coin value.

Example – the host wants to inform the hopper that 10 x 0.50 EUR coins have been added;
Version 5 protocol:

```
34 0A 00 32 00
```

Version 6 protocol:

```
34 0A 00 32 00 00 00 45 55 52
```

HALT PAYOUT 0x38:
A command the stop the current payout.

FLOAT AMOUNT 0x3D:
A command to 'float' the hopper unit to leave a request value of coins, with a requested minimum possible payout level. All coins not required to meet float value are routed to cashbox. Using protocol version 6, the host also sends a pre-test option byte (TEST_FLOAT_AMOUT 0x19, FLOAT_AMOUNT 0x58), which will determine if the command amount is tested or floated. This is useful for multi-payout systems so that the ability to pay a split down amount can be tested before committing to actual float.

Example – To float hopper unit to 100.00 EUR with a minimum a payout of 0.50 EUR.
Protocol version 5 to float to amount:

```
3D 32 00 27 10 00 00
```

Protocol version 6 to pre-test the ability to float to this amount:

```
3D 32 00 27 10 00 00 45 55 52 19
```

Protocol version 6 to float to this amount:

```
3D 32 00 27 10 00 00 45 55 52 58
```

If the float is possible the hopper will reply with generic response OK. If the float is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Hopper	0x01
Cant float exact amount	0x02
Smart Hopper Busy	0x03
Smart Hopper Disabled	0x04

GET MINIMUM PAYOUT 0x3E:

A command to payout the request the minimum level of payout available in this device.

Example:
Protocol 5 version:
3E - returns the min payout of the device.
Protocol 6 version:
3E 45 55 52 - returns the min payout of EUR in this device

SET COIN MECH INHIBITS 0x40:

This command is used to enable or disable acceptance of individual coin values from a coin acceptor connected to the hopper. The command is four bytes, the first byte is the command followed by the intended state of the inhibit (0x01 Coin acceptance enabled, 0x00 Coin acceptance disabled). The next two bytes are the value of the coin to be inhibited or enabled. In protocol version 6, the host sends the county code of the value to be inhibited or enabled as 3 byte ASCII.

Example:
To enable coin mech accept of 0.20 EUR
Protocol 5 version
40 01 14 00
Protocol 6 version
40 01 14 00 45 55 52

Any values not supported by the Smart Hopper discrimination system will be inhibited automatically, regardless of this command state.

PAYOUT BY DENOMINATION 0x46: (Version 6 or greater protocol only).

A command to payout the requested quantity of individual denominations of coins.
The quantities of coins to pay are sent as a 2 byte little endian array; the coin values as 4-byte little endian array and the country code as a 3-byte ASCII array. The host also adds an option byte to the end of the command array (TEST_PAYOUT_AMOUT 0x19 or PAYOUT_AMOUNT 0x58). This will allow a pre-test of the ability to payout the requested levels before actual payout executes.

The command is formatted as follows:

byte 0	command header
byte 1	the number of level requests (n) in this command (max 20)
byte 2 to byte 2 + (9*n)	the individual level requests (see description)
byte 2 + (9*n) + 1	the option byte, 0x19 to test this payout, 0x58 to run payout.

Individual level requests:

byte 0	- the desired level of payout (2 byte little endian)
byte 1	
byte 2	- the denomination value (4 byte little endian)
byte 3	
byte 4	
byte 5	
byte 6	- the country code of the denomination (3 byte ASCII)
byte 7	
byte 8	

Example – A hopper unit has stored 100 x 0.10 EUR, 50 x 0.20 EUR, 30 x 1.00 EUR, 10 x 1.00 GBP, 50 x 0.50 GBP and the host wishes to payout to 5 x 1.00 EUR, 5 x 0.10 EUR, 3 x 1.00 GBP and 2 x 0.50 GBP.

46 04 05 00 64 00 00 00 45 55 52 05 00 0A 00 00 00 45 55 52 03 00 64 00 00 00 47 42 50 02 00 32 00 00 00 47 42 50 19 – will test the hoppers' ability to payout these levels.

46 04 05 00 64 00 00 00 45 55 52 05 00 0A 00 00 00 45 55 52 03 00 64 00 00 00 47 42 50 02 00 32 00 00 00 47 42 50 58 – will start the payout function of these levels.

If the payout is possible the payout will reply with generic response OK. If the payout is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Hopper	0x01
Cant pay exact amount	0x02
Hopper Busy	0x03
Hopper Disabled	0x04

FLOAT BY DENOMINATION 0x44: (Version 6 or greater protocol only).

A command to float (leave in hopper) the requested quantity of individual denominations of coins.
The quantities of coins to float are sent as a 2 byte little endian array; the coin values as 4-byte little endian array and the country code as a 3-byte ASCII array. The host also adds an option byte to the end of the command array (TEST_FLOAT_AMOUT 0x19 or FLOAT_AMOUNT 0x58). This will allow a pre-test of the ability to float to the requested levels before actual float

The command is formatted as follows:

byte 0	command header
byte 1	the number of level requests (n) in this command (max 20)
byte 2 to byte 2 + (9*n)	the individual level requests (see description)
byte 2 + (9*n) + 1	the option byte, 0x19 to test this float, 0x58 to run float.

Individual level requests:
byte 0
byte 1 - the desired level to float to (2 byte little endian)
byte 2
byte 3
byte 4
byte 5 - the denomination value (4 byte little endian)
byte 6
byte 7
byte 8 - the country code of the denomination (3 byte ASCII)

Example – A hopper unit has stored 100 x 0.10 EUR, 50 x 0.20 EUR, 30 x 1.00 EUR, 10 x 1.00 GBP, 50 x 0.50 GBP and the host wishes to float to 5 x 1.00 EUR, 5 x 0.10 EUR, 3 x 1.00 GBP and 2 x 0.50 GBP.

44 04 05 00 64 00 00 00 45 55 52 05 00 0A 00 00 00 45 55 52 03 00 64 00 00 00 47 42 50 02 00 32 00 00 00 47 42 50 19 – will test the hoppers' ability to payout these levels.

44 04 05 00 64 00 00 00 45 55 52 05 00 0A 00 00 00 45 55 52 03 00 64 00 00 00 47 42 50 02 00 32 00 00 00 47 42 50 58 – will start the payout function of these levels.

To float all denominations to 0 level, send 0 as the number of requests followed by FLOAT_AMOUNT (0x58) command. The country code bytes are not necessary in this case.

If the float is possible the hopper will reply with generic response OK. If the float is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Hopper	0x01
Cant float exact amount	0x02
Smart Hopper Busy	0x03
Smart Hopper Disabled	0x04

SET COMMAND CALIBRATION 0x47 (Protocol version 6 command):
This command will set the hopper calibration mode to either AUTO_CALIBRATION or COMMAND_CALIBRATION mode.
AUTO_CALIBRATION mode (0x00) is the default state at power-up and will run the device calibration sequence at intervals it requires.
In COMMAND_CALIBRATION mode (0x01), the hopper will only run its calibration sequence when commanded to by the host. If the hopper has been set to this mode and the calibration period has timed-out, the hopper will respond with a Calibration Fail event with error byte value 7 until the RUN CALIBRATION (0x48) command is sent. This command will return 2 bytes (little endian) which gives the minimum time period in seconds between calibrations for the target Smart Hopper.

RUN COMMAND CALIBRATION 0x48 (Protocol version 6 command):
This command will run the Smart Hopper calibration sequence if the Command calibration mode has been set by command above (0x47). This command should only be sent during device idle – if the command is sent at any other time, the device will respond with Generic response COMMAND_CANNOT_BE_PROCESSED (0xF5) and error byte 3 (Busy).
If this command is sent and the COMMAND_CALIBRATION mode has not been set the response will be Parameter out of range (0xF4)

EMPTY ALL 0x3F:
This command will route all detect coins to the cash box without reporting any value and reset all the stored coin counters to zero. See SMART Empty command (0x52) to record the value emptied.

SET HOPPER OPTIONS 0x50: (Protocol version 6 only and Firmware version >= 6.04 only)

The host can set the following options for the Smart Hopper. These options do not persist in memory and after a reset they will go to their default values.

REG 0	Parameter
Bit 0	Pay Mode 1 = Free pay, 0 = Split by highest value
Bit 1	Level Check 0 = Level check disabled, 1 = level check enabled
Bit 2	Motor Speed 0 = low speed, 1 = high speed
Bit 3	Not used – set to 0
Bit 4	Not used – set to 0
Bit 5	Not used – set to 0
Bit 6	Not used – set to 0
Bit 7	Not used – set to 0

REG 1	Parameter
Bit 0	Not used – set to 0
Bit 1	Not used – set to 0
Bit 2	Not used – set to 0
Bit 3	Not used – set to 0
Bit 4	Not used – set to 0
Bit 5	Not used – set to 0
Bit 6	Not used – set to 0
Bit 7	Not used – set to 0

Two data bytes are sent, REG 0 first.

Bit 0 – Pay mode.
Split by highest value (0)– The device will attempt to payout a requested value by starting from the highest to the lowest coins available. This mode will payout the minimum number of coins possible.
Free pay (1) (Default state after reset) – the device will payout a coin as it passes its discriminator system if it fits into the current payout value and will leave enough of other coins to payout the rest of the value. This may give a faster payout but could result in a large number of coins of small denominations paid out.

Bit 1 – Level Check
Disabled (0) – the device will not refer to the level counters when calculating if a payout value can be made.
Enabled (Default state after reset) - (1) – The device will check the level counters and accept or refuse a payout request based on levels and/or split of available levels.

Bit 2 – Motor Speed.
Low speed (0) - payouts run at a lower motor speed.
High Speed (Default state after reset) (1) – The motors run at max speed for payouts.

The default values are pre-configured in the Hopper using the Hopper manager tools and can be changed off-line by the user. The defaults mentioned here are the factory supplied defaults.

GET HOPPER OPTIONS (0x51) **(Protocol version 6 only and Firmware version >= 6.04 only)**

This command returns the two options bytes detailed above.

SET COIN MECH GLOBAL INHIBIT (0x49) **(Firmware version >= 6.05 only)**

Allows the host to enable/disable the attached coin mech in one command rather than by each individual value with previous firmware versions.
Send this command and one Mode data byte:

Data byte = 0x00 – mech disabled.
Date byte = 0x01 - mech enabled.

SMART EMPTY (0x52)

Empties SMART Hopper of contents, maintaining a count of value emptied. All coin counters will be set to 0 after running this command. Use 'cashbox payout operation data' command to retrieve a breakdown of the denomination routed to the cashbox through this operation.

CASHBOX PAYOUT OPERATION DATA (0x53)

Can be sent at the end of a SMART Empty, float or dispense operation. Returns the amount emptied to cashbox from the Hopper in the last dispense, float or empty command. The quantity of denominations in the response is sent as a 2 byte little endian array; the note values as 4-byte little endian array and the country code as a 3-byte ASCII array. Each denomination in the dataset will be reported, even if 0 coins of that denomination are emptied.
As coins are emptied from the Hopper, the value is checked. An additional 4 bytes will be added to the response giving a count of object that could not be validated whilst performing the operation.

The response is formatted as follows:

byte 0	The number denominations (n) in this response (max 20)
byte 1 to byte 1 + (9*n)	The individual denomination level (see description below)
byte 1 to byte 1 + (9*n) + 1 to byte 1 to byte 1 + (9*n) + 4	The number of un-validated objects moved.

Individual level requests:

byte 0	(2 byte little endian)
byte 1	number of coins of this denomination moved to cashbox in operation
byte 2	
byte 3	
byte 4	
byte 5	denomination value (4 byte little endian)
byte 6	
byte 7	
byte 8	the country code of the denomination (3 byte ASCII)

6.9.3 SMART HOPPER RESPONSES TO POLLS

Depending on the protocol version used, the responses to Smart Hopper polls are:
PROTOCOL VERSION 5

Event/ State	Event Code
Dispensing	0xDA, Current value dispensed
Dispensed	0xD2, value dispensed
Coins Low	0xD3
Empty	0xD4
Jammed	0xD5, value dispensed
Halted	0xD6, value dispensed
Floating	0xD7, value to cashbox
Floated	0xD8, value to cashbox
Time Out	0xD9, value dispensed
Incomplete Payout	0xDC, value dispensed, value requested
Incomplete Float	0xDD, value to cashbox, value requested
CashBox Paid	0xDE, value to cashbox
Coin Credit	0xDF, value received
Coin mech jammed	0xC4
Coin mech return button pressed	0xC5
Emptying	0xC2
Emptied	0xC3
Fraud Attempt	0xE6, value paid
SMART Emptying	0xB3
SMART Emptied	0xB4

Table 23a – SMART Hopper Responses to Polls (SSP Version 5)

Dispensing: Five-byte response the last four bytes are the value of coins that have been dispensed at the point when the poll was received.
Dispensed: Five -byte response that indicates when the hopper has finished a dispense operation; the last four bytes are the value of coins that have been dispensed.
Coins Low and Empty responses were not implemented in this Smart Hopper Firmware.
Jammed: Five byte response that indicates that the hopper is jammed; this is reported until it is un-jammed or reset. It will also become disabled. The last four bytes are the value of coins that have been dispensed before the jam.
Time Out: This is given if a search for a coin fails after a time-out period and there is no way to pay that value with any others - the event will be given with 4 bytes showing the value paid out up to the time out point.
Incomplete Payout / Float: This event is given when the hopper starts up if a payout or float operation was in progress when the power was removed. The first four bytes after the event code are the value that was dispensed; the next four are the value that was originally requested.
Cashbox Paid: This event is given when coins routed to the cashbox are paid to the cashbox during a normal payout operation. The four bytes after the event code are the value routed to the cashbox.
Coin Credit: This event is given when a coin acceptor connected to the smart hopper has accepted a coin. The two bytes after the event code are the value of the coin accepted.
Coin mech jammed: This event is given when a connected coin mech reports a coin jammed in its path.

Coin mech return button pressed: An event to report when the connected coin mech had its return mechanism activated.
Fraud Attempt: This will be reported if an attempt has been made to remove coins from the hopper. The following 4 bytes give the value dispensed before the fraud was detected.
Emptying: This event is given while the Hopper is being emptied of coins into the cashbox by the EMPTY command.
Empty: This event is given at the end of the empty process.
SMART Emptying: This event is given while the hopper is being emptied of coins into the cashbox by the SMART Empty command.
SMART Emptied: This event is given at the end of the SMART empty process and is followed by4 bytes containing the value paid to the cashbox during the operation.

PROTOCOL VERSION >= 6

Event/ State	Event Code
Dispensing	0xDA
Dispensed	0xD2
Lid Open	0x81
Lid Closed	0x82
Calibration Fail	0x83
Jammed	0xD5
Halted	0xD6
Floating	0xD7
Floated	0xD8
Time Out	0xD9
Incomplete Payout	0xDC
Incomplete Float	0xDD
Emptying	0xC2
Empty	0xC3
Cash Box Paid	0xDE
Coin Credit	0xDF
Coin mech jammed	0xC4
Coin mech return button pressed	0xC5
Fraud Attempt	0xE6
Low Payout Level	0xB2
SMART Empting	0xB3
SMART Emptied	0xB4

Table 23b – SMART Hopper Responses to Polls (SSP Version 6)

Event Name: DISPENSING
Event Code: 0xDA
Event Function: Indicates that the device is in the process of paying-out a requested value. The Event will give the amounts paid out at the time of the poll for each of the countries in the device dataset. A country amount not requested will always give a zero response.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DA
1	Number of countries	1	02
2 – 5	Value dispensing country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensing country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: DISPENSED
Event Code: 0xD2
Event Function: Indicates that the device has finished paying out a requested amount. In the case of a multi-currency request, this event is not given until the full value for all currencies is paid.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D2
1	Number of countries	1	02
2 – 5	Value dispensed country 1	4	A8 02 00 00 (6.80)
6 – 8	Code country 1	3	45 55 52 ("EUR")
9 – 12	Value dispensed country 2	4	B6 03 00 00 (9.50)
13 – 15	Code country 2	3	47 42 50 ("GBP")

This example is for a two-country dataset

Event Name: FLOATING
Event Code: 0xD7
Event Function: Indicates that the device is in the process of floating to a requested value. The Event will give the amounts paid out at the time of the poll for each of the countries in the device dataset. A country amount not requested will always give a zero response.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D7
1	Number of countries	1	02
2 – 5	Value to cashbox country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 ("EUR")
9 – 12	Value to cashbox country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 ("GBP")

This example is for a two-country dataset

Event Name: FLOATED
Event Code: 0xD8
Event Function: Indicates that the device has finished floating out a requested amount. In the case of a multi-currency request, this event is not given until the full value for all currencies is paid.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D8
1	Number of countries	1	02
2 – 5	Value to cashbox country 1	4	A8 02 00 00 (6.80)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value cashbox country 2	4	B6 03 00 00 (9.50)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: LID OPEN
Event Code: 0x81
Event Function: This is reported when the Hopper detects that the top lid is not correctly engaged and will not allow any payout operations until then lid has been replaced and this event has cleared.

Event Name: LID CLOSED
Event Code: 0x82
Event Function: This is reported when the lid is replaced after being detected as open.

Event Name: CALIBRATION FAIL
Event Code: 0x83
Event Function: This is reported when the hopper has failed to correctly calibrate its sensors to the specified levels. This may be caused by an electronic failure or excessive contamination of the optical sensors. A second byte is also transmitted here giving an indication of the reason for failure:

Code	Reason
0	NO FAILURE
1	OPTICAL SENSOR FLAP
2	OPTICAL SENSOR EXIT
3	COIL SENSOR 1
4	COIL SENSOR 2
5	UINT NOT INITIALISED
6	DATA CHECKSUM ERROR
7	RE-CALIBRATION BY COMMAND REQUIRED

Table showing calibration fail error codes

Event Name: JAMMED
Event Code: 0xD5
Event Function: Indicates that the device has jammed and is unable to continue with or process any further payout requests until the jam has cleared and the device reset. The poll response reports the amount of all countries in the dataset paid out at the point of the jam.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D5
1	Number of countries	1	02
2 – 5	Value dispensed at jam country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at jam country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: HALTED
Event Code: 0xD6
Event Function: This event is given if the device has been halted by a host command or by lid removal. The values of money dispensed at the time of the halt for each country in the dataset is given.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D6
1	Number of countries	1	02
2 – 5	Value dispensed at halt country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at halt country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: TIME-OUT
Event Code: 0xD9
Event Function: Indicates that the device has timed-out while trying to payout a request. The poll response reports the amount of all countries in the dataset paid out at the point of the time-out.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D9
1	Number of countries	1	02
2 – 5	Value dispensed at time-out country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at time-out country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: INCOMPLETE PAYOUT
Event Code: 0xDD
Event Function: This event is given after a power-up when a payout of the device was interrupted by a power-down. The poll response contains the value paid out and requested for each of the counties in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DD
1	Number of countries	1	02
2 – 5	Value dispensed at power down country 1	4	DC 00 00 00 (2.20)
6 - 9	Value requested at power-down country 1	4	EE 02 00 00 (7.50)
10– 12	Code country 1	3	45 55 52 (“EUR”)
13 – 16	Value dispensed at power down country 2		6C 02 00 00 (6.20)
17 – 20	Value requested at power-down country 2	4	DE 03 00 00 (9.90)
21 – 23	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: INCOMPLETE FLOAT
Event Code: 0xDC
Event Function: This event is given after a power-up when a float of the device was interrupted by a power-down. The poll response contains the value paid out and requested for each of the counties in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DC
1	Number of countries	1	02
2 – 5	Value dispensed at power down country 1	4	DC 00 00 00 (2.20)
6 - 9	Value requested at power-down country 1	4	EE 02 00 00 (7.50)
10– 12	Code country 1	3	45 55 52 (“EUR”)
13 – 16	Value dispensed at power down country 2		6C 02 00 00 (6.20)
17 – 20	Value requested at power-down country 2	4	DE 03 00 00 (9.90)
21 – 23	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: CASHBOX PAID
Event Code: 0xDE
Event Function: This event shows the value of coins set to be routed to the cashbox that were paid to the cashbox for each country in the dataset
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DE
1	Number of countries	1	02
2 – 5	Value dispensed into cashbox country 1	4	DC 00 00 00 (2.20)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed into cashbox country 2	4	6C 02 00 00 (6.20)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: COIN CREDIT
Event Code: 0xDF
Event Function: This event shows the value of coins paid into the hopper via the connected ccTalk coin mechanism.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DF
1 – 4	Value into Hopper store.	4	64 00 00 00 (1.00)
5– 7	Code country of coin	3	45 55 52 (“EUR”)

Event Name: COIN MECHANISM JAMMED
Event Code: 0xC4
Event Function: This event is given when the attached ccTalk coin mechanism is detected as jammed.

Event Name: COIN MECHANISM RETURN ACTIVE
Event Code: 0xD5
Event Function: An event to report when the connected coin mechanism had its return route activated.

Event Name: FRAUD ATTEMPT DETECTED
Event Code: 0xE6
Event Function: Indicates that the device has detected an attempt to tamper with the hopper during idle or a payout process. The values of money paid out at the fruad detection point is given for each country in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	E6
1	Number of countries	1	02
2 – 5	Value dispensed at fraud country 1	4	A8 02 00 00 (6.80)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at fraud country 2	4	B6 03 00 00 (9.50)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: EMPTYING
Event Code: 0xC2
Event Function: This event is given while the Hopper is being emptied of coins into the cashbox by the EMPTY command. No values are reported and all detected coins are paid out.

Event Name: EMPTIED
Event Code: 0xC3
Event Function: This event is given at the end of the empty process. After emptying, all the level counters in the device will be zeroed.

Event Name: LOW PAYOUT LEVEL
Event Code: 0xB2
Event Function: This event is given when the number of coins in the Hopper store reaches a low level.

Event Name: SMART Emptying
Event Code: 0xB3
Event Function: Reported during the SMART Empty operation. Contains the value currently emptied to cashbox.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	B3
1	Number of countries	1	02
2 – 5	Value currently emptied country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value currently emptied country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: SMART Emptied
Event Code: 0xB4
Event Function: Reported after the SMART Empty operation completes. Contains the total value emptied to cashbox during the operation.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	B4
1	Number of countries	1	02
2 – 5	Value emptied to cashbox country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value emptied to cashbox country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

6.10 SMART PAYOUT

6.10.1 SMART PAYOUT OPERATION

The Smart Payout is an extension of a banknote validator, all commands are sent to the validator using its address (0x00). Information on the types of note that can be handled is obtained from the standard note validator commands.

Note that payout values are in terms of the of the penny value of that currency. So for 5.00, the value sent and returned by the hopper would be 500.

The host simply has to tell the unit the value it wishes to dispense. The unit will manage which notes are stored to be used for payout and their location to minimise the payout time, and which notes, of the type enable for storage, are sent to the stacker. This is the recommended mode of operation.

6.10.2 SMART PAYOUT COMMANDS

Action	Command code (HEX)	Data 1	Data 2	Data 3	Data 4
Enable Payout Device	0x5C	-	-	-	-
Disable Payout Device	0x5B	-	-	-	-
Set Routing	0x3B	Route, 1 byte	Value, 4 bytes	*Country code, 3 Bytes*	-
Get Routing	0x3C	Value, 4 bytes	*Country code, 3bytes*	-	-
Payout Amount	0x33	Value, 4 bytes	*Country code, 3 Bytes*	*Option, 1 byte*	-
Get Note amount	0x35	Value, 4 Bytes	*Country code, 3 bytes*	-	-
Halt Payout	0x38	-	-	-	-
Float Amount	0x3D	Min payout, 4 bytes	Value, 4 bytes	*Country Code, 3 Bytes*	*Option, 1 byte*
Get Minimum Payout	0x3E	*Country Code, 3 bytes*	-	-	-
Payout by denomination	0x46	see detail.			
Float by denomination	0x44	see detail:			
Empty All	0x3F				
SMART empty	0x52				
Cashbox Payout Operation Data	0x53	See detail			

Important note about version 6 protocol:
If the Payout unit supports protocol version 6 (Use the SET_UP_REQUEST 0x05 command to determine version) and the host has enabled protocol version 6 commands and events using the generic command HOST_PROTOCOL_VERSION 0x06, the host can send the additional parameters for country code and option to use the multi-currency and payout test amount facilities in this updated protocol.

Notice about sending note values:
All note values to and from the payout are formatted as 4 byte integer values. For example to send a note value of 250.00, we get the full value (x100) = 25000, convert this to hex = 61A8. We then send this as a 4 byte little endian order array – A8 61 00 00

ENABLE PAYOUT DEVICE 0x5C:
A command to enable the attached payout device for storing/paying out notes. A successful enable will return OK, If there is a problem the reply will be generic response COMMAND_CANNOT_BE_PROCESSED, followed by an error code:

Error reason	Error code (HEX)
No Smart Payout connected	0x01
Invalid Currency	0x02
Smart Payout Device Error	0x03

DISABLE PAYOUT DEVICE 0x5B: Single-byte command. All accepted notes will be routed to the stacker and payout commands will not be accepted.

HALT PAYOUT 0x38:
A command the stop the current payout.

SET ROUTING 0x3B:
A command to setup the desired route of a note entered into the payout unit.
Example:
To send 5.00 EUR notes to the payout store:
Version 4 protocol:
3B 00 F4 01 00 00
Version 6 protocol:
3B 00 F4 01 00 00 45 55 52

Route	code (HEX)
Note recycled and used for payouts	0x00
Note routed to cashbox	0x01

By default all note values are routed to the stacker.

GET ROUTING 0x3C:
A command to determine the route setting of a note to be entered into the payout unit.
Example:
To read the route of a 5.00 EUR note:
Version 4 protocol:
3C F4 01 00 00
Version 6 protocol:
3C F4 01 00 00 45 55 52

The command will return the route as a single byte value as below:

Route	code (HEX)
Note recycled and used for payouts	0x00
Note routed to cashbox	0x01

PAYOUT AMOUNT 0x33:

A command to set the monetary value to be paid by the payout unit. Using protocol version 6, the host also sends a pre-test option byte (TEST_PAYOUT_AMOUT 0x19, PAYOUT_AMOUNT 0x58), which will determine if the command amount is tested or paid out. This is useful for multi-payout systems so that the ability to pay a split down amount can be tested before committing to actual payout.

Example:
To payout 25.00 EUR:
Version 4 protocol to payout

```
33 C4 09 00 00
```

Version 6 Protocol to test payout ability

```
33 C4 09 00 00 45 55 52 19
```

Version 6 Protocol to payout

```
33 C4 09 00 00 45 55 52 58
```

If the payout is possible the payout will reply with generic response OK. If the payout is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Payout	0x01
Cant pay exact amount	0x02
Smart Payout Busy	0x03
Smart Payout Disabled	0x04

GET NOTE AMOUNT 0x35:

A command to return the amount of notes stored of a given denomination in the payout unit.
Example – To get the number of 5.00 EUR notes stored
Version 4 protocol:

```
35 F4 01 00 00
```

Version 6 protocol:

```
35 F4 01 00 00 45 55 52
```

FLOAT AMOUNT 0x3D:

A command to ‘float’ the payout unit to leave a request value of notes, with a requested minimum possible payout level. Using protocol version 6, the host also sends a pre-test option byte (TEST_FLOAT_AMOUT 0x19, FLOAT_AMOUNT 0x58), which will determine if the command amount is tested or floated. This is useful for multi-payout systems so that the ability to pay a split down amount can be tested before committing to actual float.

Example – To float payout unit to 250.00 EUR with a minimum a payout of 5.00 EUR.
Protocol version 4 to float to amount:

```
3D F4 01 00 00 A8 61 00 00
```

Protocol version 6 to pre-test the ability to float to this amount:

```
3D F4 01 00 00 A8 61 00 00 45 55 52 19
```

Protocol version 6 to float to this amount:

```
3D F4 01 00 00 A8 61 00 00 45 55 52 58
```

If the float is possible the payout will reply with generic response OK. If the float is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Payout	0x01
Cant float exact amount	0x02
Smart Payout Busy	0x03
Smart Payout Disabled	0x04

GET MINIMUM PAYOUT 0x3E:

A command to payout the request the minimun level of payout avialble in this device.

Example:
Protocol 4 version:

```
3E              - returns the min payout of the device.
```

Protocol 5 version:

```
3E 45 55 52     -  returns the min payout of EUR in this device
```

PAYOUT BY DENOMINATION 0x46: (Version 6 or greater protocol only).

A command to payout the requested quantity of individual denominations of notes.
The quantities of notes to pay are sent as a 2 byte little endian array; the note values as 4-byte little endian array and the country code as a 3-byte ASCII array. The host also adds an option byte to the end of the command array (TEST_PAYOUT_AMOUT 0x19 or PAYOUT_AMOUNT 0x58). This will allow a pre-test of the ability to payout the requested levels before actual payout executes.

The command is formatted as follows:

byte 0	command header
byte 1	the number of level requests (n) in this command (max 20)
byte 2 to byte 2 + (9*n)	the individual level requests (see description)
byte 2 + (9*n) + 1	the option byte, 0x19 to test this payout, 0x58 to run payout.

Individual level requests:

byte 0	- the desired level of payout (2 byte little endian)
byte 1	
byte 2	- the denomination value (4 byte little endian)
byte 3	
byte 4	
byte 5	
byte 6	- the country code of the denomination (3 byte ASCII)
byte 7	
byte 8	

Example – A payout unit has stored 20 x 5.00 EUR, 10 x 10.00 EUR, 5 x 20.00 EUR, 10 x 5.00 GBP, 5 x 10.00 GBP and the host wishes to payout to 5 x 5.00 EUR, 5 x 10.00 EUR, 3 x 5.00 GBP and 2 x 10.00 GBP.

46 04 05 00 F4 01 00 00 45 55 52 05 00 E8 03 00 00 45 55 52 03 00 F4 01 00 00 47 42 50 02 00 E8 03 00 00 47 42 50 19 – will test the payouts ability to payout these levels.

46 04 05 00 F4 01 00 00 45 55 52 05 00 E8 03 00 00 45 55 52 03 00 F4 01 00 00 47 42 50 02 00 E8 03 00 00 47 42 50 58 – will start the payout function of these levels.

If the payout is possible the payout will reply with generic response OK. If the payout is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Payout	0x01
Cant pay exact amount	0x02
Payout Busy	0x03
Payout Disabled	0x04

FLOAT BY DENOMINATION 0x44: (Version 6 or greater protocol only).

A command to float (leave in payout) the requested quantity of individual denominations of notes.
The quantities of notes to float are sent as a 2 byte little endian array; the note values as 4-byte little endian array and the country code as a 3-byte ASCII array. The host also adds an option byte to the end of the command array (TEST_FLOAT_AMOUT 0x19 or FLOAT_AMOUNT 0x58). This will allow a pre-test of the ability to float to the requested levels before actual float

The command is formatted as follows:

byte 0	command header
byte 1	the number of level requests (n) in this command (max 20)
byte 2 to byte 2 + (9*n)	the individual level requests (see description)
byte 2 + (9*n) + 1	the option byte, 0x19 to test this float, 0x58 to run float.

Individual level requests:
byte 0
byte 1 - the desired level to float to (2 byte little endian)
byte 2
byte 3
byte 4
byte 5 - the denomination value (4 byte little endian)
byte 6
byte 7
byte 8 - the country code of the denomination (3 byte ASCII)

Example – A payout unit has stored 20 x 5.00 EUR, 10 x 10.00 EUR, 5 x 20.00 EUR, 10 x 5.00 GBP, 5 x 10.00 GBP and the host wishes to float to 5 x 5.00 EUR, 5 x 10.00 EUR, 3 x 5.00 GBP and 2 x 10.00 GBP.

44 04 05 00 F4 01 00 00 45 55 52 05 00 E8 03 00 00 45 55 52 03 00 F4 01 00 00 47 42 50 02 00 E8 03 00 00 47 42 50 19 – will test the payouts ability to float to these levels.
44 04 05 00 F4 01 00 00 45 55 52 05 00 E8 03 00 00 45 55 52 03 00 F4 01 00 00 47 42 50 02 00 E8 03 00 00 47 42 50 58 – will start the float function to these levels.

If the float is possible the payout will reply with generic response OK. If the float is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Not enough value in Smart Payout	0x01
Cant float exact amount	0x02
Smart Payout Busy	0x03
Smart Payout Disabled	0x04

EMPTY ALL 0x3F:

A Command to send all notes stored in the payout to the stacker for removal.

SMART EMPTY (0x52)

Empties SMART Payout of notes, maintaining a count of value emptied. All note counters will be set to 0 after running this command. Use 'cashbox payout operation data' command to retrieve a breakdown of the denomination routed to the cashbox through this operation.

CASHBOX PAYOUT OPERATION DATA (0x53)

Can be sent at the end of a SMART Empty, float or dispense operation. Returns the amount emptied to cashbox from the Payout in the last dispense, float or empty command. The quantity of denominations in the response is sent as a 2 byte little endian array; the note values as 4-byte little endian array and the country code as a 3-byte ASCII array. Each denomination in the dataset will be reported, even if 0 notes of that denomination are emptied.
As notes are removed from the storage unit destined for cashbox or payout, they are re-validated. An additional 4 bytes will be added giving a count of notes that could not be validated whilst performing the operation.

The response is formatted as follows:

byte 0	The number denominations (n) in this response (max 20)
byte 1 to byte 1 + (9*n)	The individual denomination level (see description below)
byte 1 to byte 1 + (9*n) + 1 to byte 1 to byte 1 + (9*n) + 4	The number of un-validated notes moved.

Individual level requests:

byte 0	(2 byte little endian)
byte 1	number of notes of this denomination moved to cashbox in operation
byte 2	
byte 3	
byte 4	
byte 5	denomination value (4 byte little endian)
byte 6	
byte 7	
byte 8	the country code of the denomination (3 byte ASCII)

6.10.3 SMART PAYOUT RESPONSE TO POLLS

These responses to polls are in addition to the BNV responses and are only valid when a Smart Payout is fitted.

PROTOCOL VERSION 4

Event/ State	Event Code
Dispensing	0xDA, Current value dispensed
Dispensed	0xD2, value dispensed
Jammed	0xD5, value dispensed
Halted	0xD6, value dispensed
Floating	0xD7, value to cashbox
Floated	0xD8, value to cashbox
Time Out	0xD9, value dispensed
Incomplete Payout	0xDC, value dispensed, value requested
Incomplete Float	0xDD, value to cashbox, value requested
Emptying	0xC2
Empty	0xC3
Note stored in payout	0xDB
SMART Emptying	0xB3
SMART Emptied	0xB4

Dispensing: Five-byte response the last four bytes are the value of notes that have been dispensed at the point when the poll was received.
Dispensed: Five -byte response that indicates when the payout has finished a dispense operation; the last four bytes are the value of notes that have been dispensed.
Jammed: Five byte response that indicates that the payout is jammed; this is reported until it is un-jammed or reset. It will also become disabled. The last four bytes are the value of notes that have been dispensed before the jam.
Time Out: This is given if a search for a note in the payout store fails after a time-out period and there is no way to pay that value with any others - the event will be given with 4 bytes showing the value paid out up to the time out point.
Incomplete Payout / Float: This event is given when the payout starts up if a payout or float operation was in progress when the power was removed. The first four bytes after the event code are the value that was dispensed; the next four are the value that was originally requested.
Note stored in payout: This event is given when notes paid in to the payout system are routed to the payout store.
Emptying: This event is given while the payout is being emptied of notes into the cashbox by the EMPTY command.
Empty: This event is given at the end of the empty process.
SMART Emptying: This event is given while the payout is being emptied of notes into the cashbox by the SMART Empty command.
SMART Emptied: This event is given at the end of the SMART empty process and is followed by4 bytes containing the value paid to the cashbox during the operation.

PROTOCOL VERSION >=6

Event/ State	Event Code	Protocol version >=
Dispensing	0xDA	4
Dispensed	0xD2	4
Jammed	0xD5	4
Halted	0xD6	4
Floating	0xD7	4
Floated	0xD8	4
Time Out	0xD9	4
Incomplete Payout	0xDC	4
Incomplete Float	0xDD	4
Emptying	0xC2	4
Empty	0xC3	4
Payout out of service	0xC6	6
Note stored in payout	0xDB	4
Jam Recovery	0xB0	7
Error During Payout	0xB1	7
SMART Emptying	0xB3	4
SMART Emptied	0xB4	4
Channel Disable	0xB5	7

Notes on version 6 protocol responses:
The 3-byte country code responses in the event list will only be generated by a protocol 6 compatible payout when the host has enabled version 6 protocols using the generic HOST PROTOCOL VERSION command 0x06.

Event Name: DISPENSING
Event Code: 0xDA
Event Function: Indicates that the device is in the process of paying-out a requested value. The Event will give the amounts paid out at the time of the poll for each of the countries in the device dataset. A country amount not requested will always give a zero response.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DA
1	Number of countries	1	02
2 – 5	Value dispensing country 1	4	E8 03 00 00 (10.00)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensing country 2	4	C4 09 00 00 (25.00)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: DISPENSED
Event Code: 0xD2
Event Function: Indicates that the device has finished paying out a requested amount. In the case of a multi-currency request, this event is not given until the full value for all currencies is paid.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D2
1	Number of countries	1	02
2 – 5	Value dispensed country 1	4	D0 07 00 00 (20.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed country 2	4	A0 0F 00 00 (40.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: FLOATING
Event Code: 0xD7
Event Function: Indicates that the device is in the process of floating to a requested value. The Event will give the amounts paid out at the time of the poll for each of the countries in the device dataset. A country amount not requested will always give a zero response.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D7
1	Number of countries	1	02
2 – 5	Value to cashbox country 1	4	E8 03 00 00 (10.00)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value to cashbox country 2	4	C4 09 00 00 (25.00)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: FLOATED
Event Code: 0xD8
Event Function: Indicates that the device has finished floating out a requested amount. In the case of a multi-currency request, this event is not given until the full value for all currencies is paid.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D8
1	Number of countries	1	02
2 – 5	Value to cashbox country 1	4	D0 07 00 00 (20.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value cashbox country 2	4	A0 0F 00 00 (40.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: JAMMED
Event Code: 0xD5
Event Function: Indicates that the device has jammed and is unable to continue with or process any further payout requests until the jam has cleared and the device reset. The poll response reports the amount of all countries in the dataset paid out at the point of the jam.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D5
1	Number of countries	1	02
2 – 5	Value dispensed at jam country 1	4	E8 03 00 00 (10.00)
6– 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at jam country 2	4	C4 09 00 00 (25.00)
13 - 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: HALTED
Event Code: 0xD6
Event Function: This event is given if the device has been halted by a host command. The values of money dispensed at the time of the halt for each country in the dataset is given.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D6
1	Number of countries	1	02
2 – 5	Value dispensed at halt country 1	4	E8 03 00 00 (10.00)
6– 8	Code country 1	3	45 55 52 ("EUR")
9 – 12	Value dispensed at halt country 2	4	C4 09 00 00 (25.00)
13 - 15	Code country 2	3	47 42 50 ("GBP")

This example is for a two-country dataset

Event Name: TIME-OUT
Event Code: 0xD9
Event Function: Indicates that the device has timed-out while trying to payout a request. The poll response reports the amount of all countries in the dataset paid out at the point of the time-out.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	D9
1	Number of countries	1	02
2 – 5	Value dispensed at time-out country 1	4	E8 03 00 00 (10.00)
6– 8	Code country 1	3	45 55 52 ("EUR")
9 – 12	Value dispensed at time-out country 2	4	C4 09 00 00 (25.00)
13 - 15	Code country 2	3	47 42 50 ("GBP")

This example is for a two-country dataset

Event Name: INCOMPLETE PAYOUT
Event Code: 0xDD
Event Function: This event is given after a power-up when a payout of the device was interrupted by a power-down. The poll response contains the value paid out and requested for each of the counties in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DD
1	Number of countries	1	02
2 – 5	Value dispensed at power down country 1	4	F4 01 00 00 (5.00)
6 - 9	Value requested at power-down country 1	4	E8 03 00 00 (10.00)
10– 12	Code country 1	3	45 55 52 (“EUR”)
13 – 16	Value dispensed at power down country 2		E8 03 00 00 (10.00)
17 – 20	Value requested at power-down country 2	4	D0 07 00 00 (20.00)
21 – 23	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: INCOMPLETE FLOAT
Event Code: 0xDC
Event Function: This event is given after a power-up when a float of the device was interrupted by a power-down. The poll response contains the value paid out and requested for each of the counties in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	DC
1	Number of countries	1	02
2 – 5	Value dispensed at power down country 1	4	F4 01 00 00 (5.00)
6 - 9	Value requested at power-down country 1	4	E8 03 00 00 (10.00)
10– 12	Code country 1	3	45 55 52 (“EUR”)
13 – 16	Value dispensed at power down country 2		E8 03 00 00 (10.00)
17 – 20	Value requested at power-down country 2	4	D0 07 00 00 (20.00)
21 – 23	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: FRAUD ATTEMPT DETECTED
Event Code: 0xE6
Event Function: Indicates that the device has detected an attempt to tamper with the payout during idle or a payout process. The values of money paid out at the fruad detection point is given for each country in the dataset.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	E6
1	Number of countries	1	02
2 – 5	Value dispensed at fraud country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at fraud country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: EMPTYING
Event Code: 0xC2
Event Function: This event is given while the Hopper is being emptied of coins into the cashbox by the EMPTY command. No values are reported and all detected coins are paid out.

Event Name: EMPTIED
Event Code: 0xC3
Event Function: This event is given at the end of the empty process. After emptying, all the level counters in the device will be zeroed.

Event Name: NOTE STORED IN PAYOUT
Event Code: 0xDB
Event Function: This event is given when notes paid in to the payout system are routed to the payout store.

Event Name: NOTE PAYOUT OUT-OF-SERVICE
Event Code: 0xDB
Event Function: This event is given if the payout goes out of service during operation. If this event is detected after a poll, the host can send the ENABLE PAYOUT DEVICE command to determine if the payout unit comes back into service.

Event Name: JAM RECOVERY
Event Code: 0xB0
Event Function: The SMART Payout unit is in the process of recovering from a detected jam. This process will typically move five notes to the cash box; this is done to minimise the possibility the unit will go out of service. Only reported if using protocol version >= 7.

Event Name: ERROR DURING PAYOUT
Event Code: 0xB1
Event Function: Returned if an error is detected whilst moving a note inside the SMART Payout unit. The cause of error (1 byte) indicates the source of the condition; 0x00 for note not being correctly detected as it is routed to cashbox or for payout, 0x01 if note is jammed in transport. In the case of the incorrect detection, the response to Cashbox Payout Operation Data request would report the note expected to be paid out. Only reported if using protocol version >= 7.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	B1
1	Number of countries	1	02
2 – 5	Value dispensed at jam country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value dispensed at jam country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)
16	Cause of error	1	00

This example is for a two-country dataset

Event Name: SMART Emptying
Event Code: 0xB3
Event Function: Reported during the SMART Empty operation. Contains the value currently emptied to cashbox.
The poll response:

Byte	Use	Size bytes	Example
0	Event Code	1	B3
1	Number of countries	1	02
2 – 5	Value currently emptied country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value currently emptied country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: SMART Emptied
Event Code: 0xB4
Event Function: Reported after the SMART Empty operation completes. Contains the total value emptied to cashbox during the operation.
The poll response:

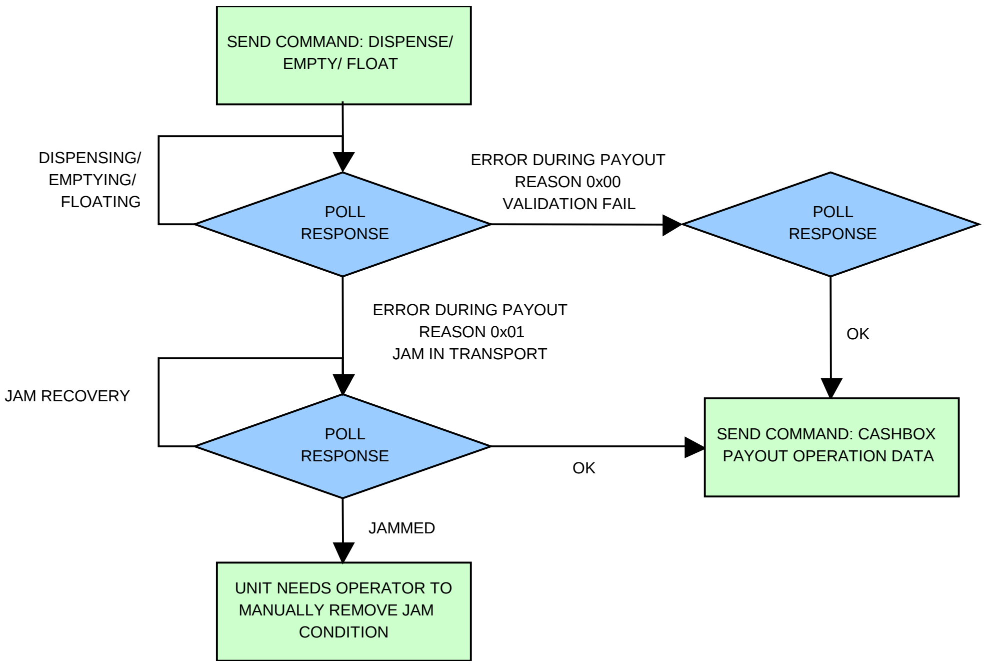
Byte	Use	Size bytes	Example
0	Event Code	1	B4
1	Number of countries	1	02
2 – 5	Value emptied to cashbox country 1	4	E8 03 00 00 (10.00)
6 – 8	Code country 1	3	45 55 52 (“EUR”)
9 – 12	Value emptied to cashbox country 2	4	C4 09 00 00 (25.00)
13 – 15	Code country 2	3	47 42 50 (“GBP”)

This example is for a two-country dataset

Event Name: Channel Disable
Event Code: 0xB5
Event Function: Indicates all note channels have been inhibited and as such, the unit is disabled. Only reported if using protocol version 7 and above.

6.10.4 ERROR DURING PAYOUT RECOVERY (PROTOCOL VERSION 7 ONLY)

If a note is detected as jammed or fails validation when it is being moved from the payout to the cashbox or to be paid back to the customer, event 0xB1 will be reported in poll events. The simplified flowchart below demonstrates the possible routes to recovery from this event, dependent on the cause.


SEND COMMAND: DISPENSE/ EMPTY/ FLOAT
DISPENSING/ EMPTYING/ FLOATING
POLL RESPONSE
ERROR DURING PAYOUT REASON 0x00 VALIDATION FAIL
POLL RESPONSE
ERROR DURING PAYOUT REASON 0x01 JAM IN TRANSPORT
OK
JAM RECOVERY
POLL RESPONSE
SEND COMMAND: CASHBOX PAYOUT OPERATION DATA
OK
JAMMED
UNIT NEEDS OPERATOR TO MANUALLY REMOVE JAM CONDITION

If the cause is validation fail, the Cashbox Payout Operation Data will only report the note expected to be validated.

In all cases, the note counters inside the SMART payout will report the number of notes remaining for payout.

6.11 NOTE FLOAT

6.11.1 NOTE FLOAT OPERATION

The Note Float is an extension of a banknote validator, all commands are sent to the validator using its address (0x00). Information on the types of note that can be handled is obtained from the standard note validator commands.

Note that by default, payout values are in terms of the of the penny value of that currency. So for 5.00, the value sent and returned by the Note Float would be 500. This can be changed to report the channel number of the note from the validator by using the Set Value Reporting Type command.

The Note Float is a LIFO (last in first out) system, this means that only the last note is available to be paid out or moved to the stacker. Any value of note can be routed into the Note Float, using the Set Routing command. In the simplest implementation only one value of note would be set to be stored. The host can only tell the Note Float to payout or stack the last note in the LIFO. In this way later notes can be paid out by stacking notes earlier in the LIFO.

In the case where more than one value of note is routed to the Note Float the host machine must manage which notes it wishes to payout or move to the stacker. Three commands are available to enable this: Get Note Positions, Payout Note and Stack Note.

6.11.2 NOTE FLOAT COMMANDS

Action	Command code (HEX)
Enable Payout Device *	0x5C, Options
Disable Payout Device	0x5B
Set Routing *	0x3B, route, value/channel (Country code)
Get Routing *	0x3C, value/channel (Country code)
Empty	0x3F
Get Note Positions *	0x41
Payout Note	0x42
Stack Note	0x43
Set Value Reporting Type *	0x45, type
SMART empty	0x52
Cashbox Payout Operation Data	0x53

* The commands marked with * will respond with the generic response Command cannot be processed and an error code of Invalid Currency if there are notes inside the Note Float that do not match the dataset that is installed in the note validator.

Enable Payout Device: Two-byte command, the first byte is the command and the second enables several different options as shown in the table below. Unused bits should be 0.

Bit	Meaning if Set
7	Reserved: Send 0
6	Reserved: Send 0
5	Reserved: Send 0
4	Reserved: Send 0
3	Reserved: Send 0
2	Reserved: Send 0
1	Reserved: Send 0
0	VALUE_ON_STORED: Enable Note value to be sent with STORED event.

If there is a problem the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
No Note Float connected	0x01
Invalid Currency	0x02
Note Float Device Error	0x03
Empty Only	0x04

Disable Payout Device: Single-byte command. All accepted notes will be routed to the stacker and payout commands will not be accepted.

Set Value Reporting Type: Two-byte command. This will set the method of reporting values of notes. There are two options, by a four-byte value of the note or by the channel number of the value from the banknote validator. If the channel number is used then the actual value must be determined using the data from the Validator command Unit Data. The default operation is by 4-byte value.

Value Reporting Type	code (HEX)
Values reported by 4 byte value	0x00
Values reported by channel number	0x01

Set Routing: The first byte is the command. The second byte is the selected route, and the remaining bytes are the four-byte value or single byte channel number (depending on the reporting type selected) of the note that the route should be applied to. By default all note values are stacked. For protocol version 6, three extra country code bytes are sent

Route	code (HEX)
Note stored and used for payouts	0x00
Note routed to cashbox	0x01

Empty: Single byte command, this will cause all notes to be sent to the stacker for removal.

Get Note Positions: Single byte command, this will return the number of notes in the Note Float and the value in each position. The way the value is reported is specified by the Set Reporting Type command. The value can be reported by its value or by the channel number of the bill validator. The first note in the table is the first note that was paid into the Note Float. The Note Float is a LIFO system, so the note that is last in the table is the only one that is available to be paid out or moved into the stacker.

If report by value is selected then the response will be as below:

Byte offset	Parameter
0	Number of notes in the Note Float (n)
1 – 4	Value of note in slot 1
5 – 8	Value of note in slot 2
9 – 12	Value of note in slot 3
....	....
(n * 4) – (n * 4)+4	Value of Note in Slot n

If report by channel is selected then the response will be as below:

Byte offset	Parameter
0	Number of notes in the Note Float (n)
1	Channel of note in slot 1
2	Channel of note in slot 2
3	Channel of note in slot 3
....	....
n	Channel of Note in Slot n

Payout Note: This is a single byte command. The Note Float will payout the last note that was stored. This is the note that is in the highest position in the table returned by the Get Note Positions Command. If the payout is possible the Note Float will reply with generic response OK. If the payout is not possible the reply will be generic response command cannot be processed, followed by an error code:

Error reason	Error code (HEX)
Note Float Not Connected	0x01
Note Float empty	0x02
Note Float Busy	0x03
Note Float Disabled	0x04

Stack Note: This is a single byte command. The Note Float will stack the last note that was stored. This is the note that is in the highest position in the table returned by the Get Note Positions Command. If the stack operation is possible the Note Float will reply with generic response OK. If the stack is not possible the reply will be generic response command cannot be processed, followed by an error code as shown above.

Smart Empty:
Empties Note Float of notes, maintaining a count of value emptied. The note counters will be set to 0 after running this command. Use 'cashbox payout operation data' command to retrieve a breakdown of the denomination routed to the cashbox through this operation.

Cashbox Payout Operation Data:

Can be sent at the end of a SMART Empty, float or dispense operation. Returns the amount emptied to cashbox from the Note Float in the last dispense, float or empty command. The quantity of denominations in the response is sent as a 2 byte little endian array; the note values as 4-byte little endian array and the country code as a 3-byte ASCII array. Each denomination in the dataset will be reported, even if 0 notes of that denomination are emptied.

The response is formatted as follows:

byte 0	The number denominations (n) in this response (max 20)
byte 1 to byte 1 + (9*n)	The individual denomination level (see description below)

Individual level requests:

byte 0	(2 byte little endian)
byte 1	number of notes of this denomination moved to cashbox in operation
byte 2	
byte 3	
byte 4	
byte 5	denomination value (4 byte little endian)
byte 6	
byte 7	
byte 8	the country code of the denomination (3 byte ASCII)

6.11.3 NOTE FLOAT RESPONSE TO POLLS

These responses to polls are in addition to the BNV responses and are only valid when a Smart Payout is fitted.

Event/ State	Event Code
Dispensing	0xDA, value/channel dispensing
Dispensed	0xD2, value/channel dispensed
Jammed	0xD5, value/channel dispensed
Halted	0xD6
Incomplete Payout	0xDD, value/channel dispensed, value/channel requested
Emptying	0xC2
Empty	0xC3
Note stored in payout	0xDB, (value/channel of note) *
Note Transferred to Stacker	0xC9, value/channel of note
Payout out of service	0xC6
Note paid into stacker at power-up	0xCA (value/channel of note)
Note paid into store at power up	0xCB (value/channel of note)
Note dispensed at power up	0xCD (value/channel of note)
Note Float Removed	0xC7
Note Float Attached	0xC8
Note in Bezel Hold	0xCE (value/channel of note)
Device Full	0xCF
SMART Emptying	0xB3
SMART Emptied	0xB4
Channel Disable	0xB5

***When enabled using option flag**

Dispensing: Single-byte response indicating that a dispense operation is in progress. The four-byte value of the note or the single byte channel number is reported, depending on the reporting type set. This value will be 0 until the note has passed out of the Note float and into a payable position in the validator.
Note In Bezel Hold: This event is reported when the Dispensing note is held in the bezel waiting for the user to remove it.
Dispensed: Response that indicates when the payout has finished a dispense operation; The four-byte value of the note or the single byte channel number is reported, depending on the reporting type set.
Jammed: Five byte response that indicates that the payout is jammed; this is reported until it is un-jammed or reset. It will also become disabled. The value or channel number of the note being dispensed is also reported.
Incomplete Payout: This event is given when the payout starts up if a payout or float operation was in progress when the power was removed. The first four bytes after the event code are the value that was dispensed; the next four are the value that was originally requested.
Note stored in payout: This event is given when notes paid in to the payout system are routed to the payout store. For compatibility with the Smart Payout the value of the note is not reported. However of the Note Float is enabled with the option flag VALUE_ON_STORED set, then the value of the note will be reported (See enable payout device command).

Emptying: This event is given while the payout is being emptied of notes into the cashbox by the EMPTY command.
Empty: This event is given at the end of the empty process.
Note Transferred to Stacker: This event is given when a note has successfully been moved from the Note Float and Stacked in the cash box. During the process the Stacking event will be given.
Note Float Removed: This event is reported when the Note Float is physically disconnected from the Note Validator while the power is on.
Note Float Attached: This event is reported when the Note Float is physically attached to the Note Validator while the power is on. The Note Validator and Note Float will then reset.
Device Full: This event is reported when the Note Float has reached its limit of stored notes. This event will be reported until a note is paid out or stacked.
Channel Disable: Indicates all note channels have been inhibited and as such, the unit is disabled. Only reported if using protocol version 7 and above.

APPENDIX A – COUNTRY CODES

Country	Abbr.	Country	Abbr.	Country	Abbr.
United Arab Emirates	AED	Georgia	GEL	Peru	PEN
Albania	ALL	Guatemala	GTQ	Philippines	PHP
Armenia	AMD	Hong Kong Mixed	H00	Poland	PLN
Angola	AOA	Hong Kong	HKD	Peru	PSS
Argentina	ARS	Honduras	HNL	Paraguay	PYG
Australia	AUD	Croatia	HRK	Qatar	QAR
Azerbaijan	AZN	Hungary	HUF	Romania	RON
Bosnia	BAM	Indonesia	IDR	Serbia	RSD
Barbados	BBD	Israel	ILS	Russia	RUB
Bangladesh	BDT	India	INR	Sweden Mixed	S00
Bulgaria	BGN	Iran	IRR	Saudi Arabia	SAR
Bahrain	BHD	Iceland	ISK	Sudan	SDG
Brunei	BND	Jamaica	JMD	Sweden	SEK
Bolivia	BOB	Jordan	JOD	Singapore	SGD
Brazil	BRL	Japan	JPY	Slovenia	SIT
Bahamas	BSD	Kenya	KES	Slovakia	SKK
Belarus	BYR	Kyrgyzstan	KGS	Syria	SYP
Canada	CAD	South Korea	KRW	Thailand	THB
Switzerland	CHF	Kuwait	KWD	Tajikistan	TJS
Chile	CLP	Kazakhstan	KZT	Thailand	TLB
China	CNY	Lebanon	LBP	Turkmenistan	TMT
Columbia	COP	Lithuania	LTL	Turkey	TRY
Costa Rica	CRC	Latvia	LVL	Taiwan	TWD
Czech Rep	CZK	Morocco	MAD	Tanzania	TZS
Denmark	DKK	Moldova	MDL	USA Mixed	U00
Dominican Rep	DOP	Macedonia	MKD	Ukraine	UAH
Euro	E00	Mongolia	MNT	Uganda	UGX
Euro Mixed (1)	EDN	Mexico	MXN	USA	USD
Estonia	EEK	Malaysia	MYR	Uzbekistan	UZS
Egypt	EGP	Norway Mixed	N00	Venezuela	VEF
Euro Mixed (2)	ESD	Namibia	NAD	Vietnam	VND
Euro Mixed (3)	EUC	Nigeria	NGN	Central Africa Franc	XAF
Euro	EUR	Nicaragua	NIO	Eastern Caribbean	XCD
Euro Mixed (4)	EUS	Norway	NOK	West Africa Franc	XOF
UK Mixed	G00	New Zealand	NZD	Yemen	YER
UK	GBP	Oman	OMR	South Africa	ZAR

APPENDIX B – BLOCK ENCRYPTION & KEY TRANSFER ROUTINES

Please contact ITL for an implementation of key exchange and AES encryption, this is provided as C source code.

The encryption functions included in issue 14 and earlier of this document should not be used.

APPENDIX C – CRC CALCULATION ROUTINES

```
#define FALSE           0x00
#define TRUE            0x01
unsigned char CRCL,CRCH;
int CRC_Table[8*32]={
      0x0000,0x8005,0x800F,0x000A,0x801B,0x001E,0x0014,0x8011,
      0x8033,0x0036,0x003C,0x8039,0x0028,0x802D,0x8027,0x0022,
      0x8063,0x0066,0x006C,0x8069,0x0078,0x807D,0x8077,0x0072,
      0x0050,0x8055,0x805F,0x005A,0x804B,0x004E,0x0044,0x8041,
      0x80C3,0x00C6,0x00CC,0x80C9,0x00D8,0x80DD,0x80D7,0x00D2,
      0x00F0,0x80F5,0x80FF,0x00FA,0x80EB,0x00EE,0x00E4,0x80E1,
      0x00A0,0x80A5,0x80AF,0x00AA,0x80BB,0x00BE,0x00B4,0x80B1,
      0x8093,0x0096,0x009C,0x8099,0x0088,0x808D,0x8087,0x0082,
      0x8183,0x0186,0x018C,0x8189,0x0198,0x819D,0x8197,0x0192,
      0x01B0,0x81B5,0x81BF,0x01BA,0x81AB,0x01AE,0x01A4,0x81A1,
      0x01E0,0x81E5,0x81EF,0x01EA,0x81FB,0x01FE,0x01F4,0x81F1,
      0x81D3,0x01D6,0x01DC,0x81D9,0x01C8,0x81CD,0x81C7,0x01C2,
      0x0140,0x8145,0x814F,0x014A,0x815B,0x015E,0x0154,0x8151,
      0x8173,0x0176,0x017C,0x8179,0x0168,0x816D,0x8167,0x0162,
      0x8123,0x0126,0x012C,0x8129,0x0138,0x813D,0x8137,0x0132,
      0x0110,0x8115,0x811F,0x011A,0x810B,0x010E,0x0104,0x8101,
      0x8303,0x0306,0x030C,0x8309,0x0318,0x831D,0x8317,0x0312,
      0x0330,0x8335,0x833F,0x033A,0x832B,0x032E,0x0324,0x8321,
      0x0360,0x8365,0x836F,0x036A,0x837B,0x037E,0x0374,0x8371,
      0x8353,0x0356,0x035C,0x8359,0x0348,0x834D,0x8347,0x0342,
      0x03C0,0x83C5,0x83CF,0x03CA,0x83DB,0x03DE,0x03D4,0x83D1,
      0x83F3,0x03F6,0x03FC,0x83F9,0x03E8,0x83ED,0x83E7,0x03E2,
      0x83A3,0x03A6,0x03AC,0x83A9,0x03B8,0x83BD,0x83B7,0x03B2,
      0x0390,0x8395,0x839F,0x039A,0x838B,0x038E,0x0384,0x8381,
      0x0280,0x8285,0x828F,0x028A,0x829B,0x029E,0x0294,0x8291,
      0x82B3,0x02B6,0x02BC,0x82B9,0x02A8,0x82AD,0x82A7,0x02A2,
      0x82E3,0x02E6,0x02EC,0x82E9,0x02F8,0x82FD,0x82F7,0x02F2,
      0x02D0,0x82D5,0x82DF,0x02DA,0x82CB,0x02CE,0x02C4,0x82C1,
      0x8243,0x0246,0x024C,0x8249,0x0258,0x825D,0x8257,0x0252,
      0x0270,0x8275,0x827F,0x027A,0x826B,0x026E,0x0264,0x8261,
      0x0220,0x8225,0x822F,0x022A,0x823B,0x023E,0x0234,0x8231,
      0x8213,0x0216,0x021C,0x8219,0x0208,0x820D,0x8207,0x0202};

//---------------------------------------------------------------
void Update_CRC(unsigned char num){
      unsigned int  table_addr;
      table_addr=(num ^ CRCH);
      CRCH=(CRC_Table[table_addr] >> 8) ^ CRCL;
      CRCL=(CRC_Table[table_addr] & 0x00FF);
}
//----------------------------------------------------------------
----------
void Reset_CRC(void){
      CRCL=0xFF;
      CRCH=0xFF;
}
```