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**	**Mod By**	**Comments**
Issue 1	1	26/05/98	PD	
Issue 2	1	03/02/99	TB	
Issue 3	1	11/06/99	AK	
Issue 4	2	4/02/00	PD	
Issue 5	2	20/06/00	PD	
Issue 6	2	26/10/00	PD	
Issue 7	2	20/11/00	PD	
Issue 8	2	20/01/01	TB	
Issue 9	2	4/10/01	TB	
Issue 10	2	21/01/02	AK	
Issue 11	2	23/03/04	PK	General Revision
Issue 12	3	05/08/04	TB	Protocol Version 3
Issue 13	3	02/01/06	TB	Protocol Version 3 events
Issue 14	4	16/11/07	TB	BNV Barcode commands
Issue 14L	4		PD	Encryption, Smart Range
Issue 14M	4	21/6/09	TB	Encryption, Smart Range
Issue 15	4	23/7/09	TB	Encryption, Smart Range Release
Issue 16	4	1/9/09	TB	Payout command mod

Document Issue 15 - Protocol Version 4

Issue 4. – Peter Dunlop 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 – Peter Dunlop 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. – Peter Dunlop 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. – Peter Dunlop 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. – Tim Beswick 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. – Tim Beswick 04/10/01
Addition of HOLD command to allow escrow implementation on BNV

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

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

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

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

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

Issue 14K – Peter Dunlop 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 – Tim Beswick 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 On SMART HOPPER changed from 3-byte to 5-byte command.

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

Issue 16 – Tim Beswick 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.

TABLE OF CONTENTS

1.0 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

Innovative Technology has a number of European and International Patents and Patents Pending protecting this product. If you require further details please contact ITL®.

INNOVATIVE TECHNOLOGY IS NOT RESPONSIBLE FOR ANY LOSS, HARM, OR DAMAGE CAUSED BY THE INSTALLATION AND USE OF THIS PRODUCT. THIS DOES NOT AFFECT YOUR LOCAL STATUTORY RIGHTS. IF IN DOUBT PLEASE CONTACT INNOVATIVE TECHNOLOGY FOR DETAILS OF ANY CHANGES

2.0 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).

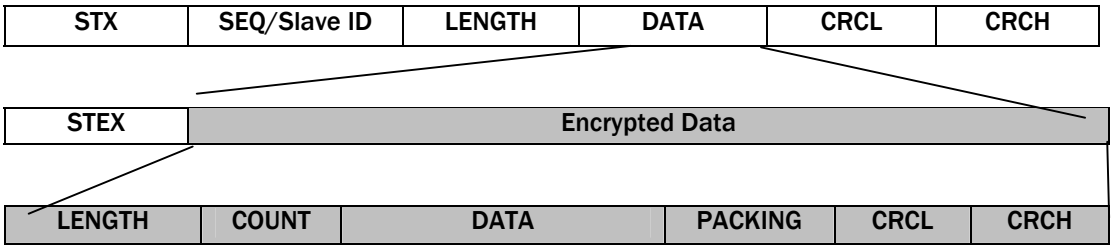

Figure 1 – Encrypted Data Format

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 5 seconds. The slave will deem the host to be inactive, if the time between polls is greater than 5 seconds. If the slave does not receive a poll within 5 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, current version is 02.

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.

6.4.4 EXAMPLE PROGRAMMING FILE FORMATS (ITL NV4 VALIDATOR).

Programming files supplied by Innovative Technology Ltd for NV4 BNV are formatted as follows (see tables 11 and 12): Variable number of blocks depending on currency, fixed block length (128 bytes).

Header block	92, 01, 49, 54, 4C, 01, 01, 01, 80, 8D, 2C	Padded to block length with 0's
1st data block	F0, 34, C0, 21, D3, 00, 00, 5F, 5F,	80h data bytes
2nd data block	FF, 24, D3, 21, 45, 01, 00, 3F, 5F,	80h data bytes
	.	
	.	
257th data block	FF, 24, D3, 21, 45, 01, 00, 3F, 5F,	80h data bytes

Table 11 - Currency File Example

Header block	01, 01, 49, 54, 4C, 00, 01, 01, 80, FD, 22	Padded to block length with 0's
1st data block	FF, 24, D3, 21, 45, 01, 00, 3F, 5F,	80h data bytes
2nd data block	F0, 34, C0, D1, D3, 00, 00, 5F, 5F	80h data bytes
	.	
	.	
257th data block	FF, 24, D3, 21, 45, 01, 00, 3F, 5F,	80h data bytes

Table 12 - Firmware File Example

6.4.5 SIMPLIFIED REMOTE PROGRAMMING FLOW CHART.

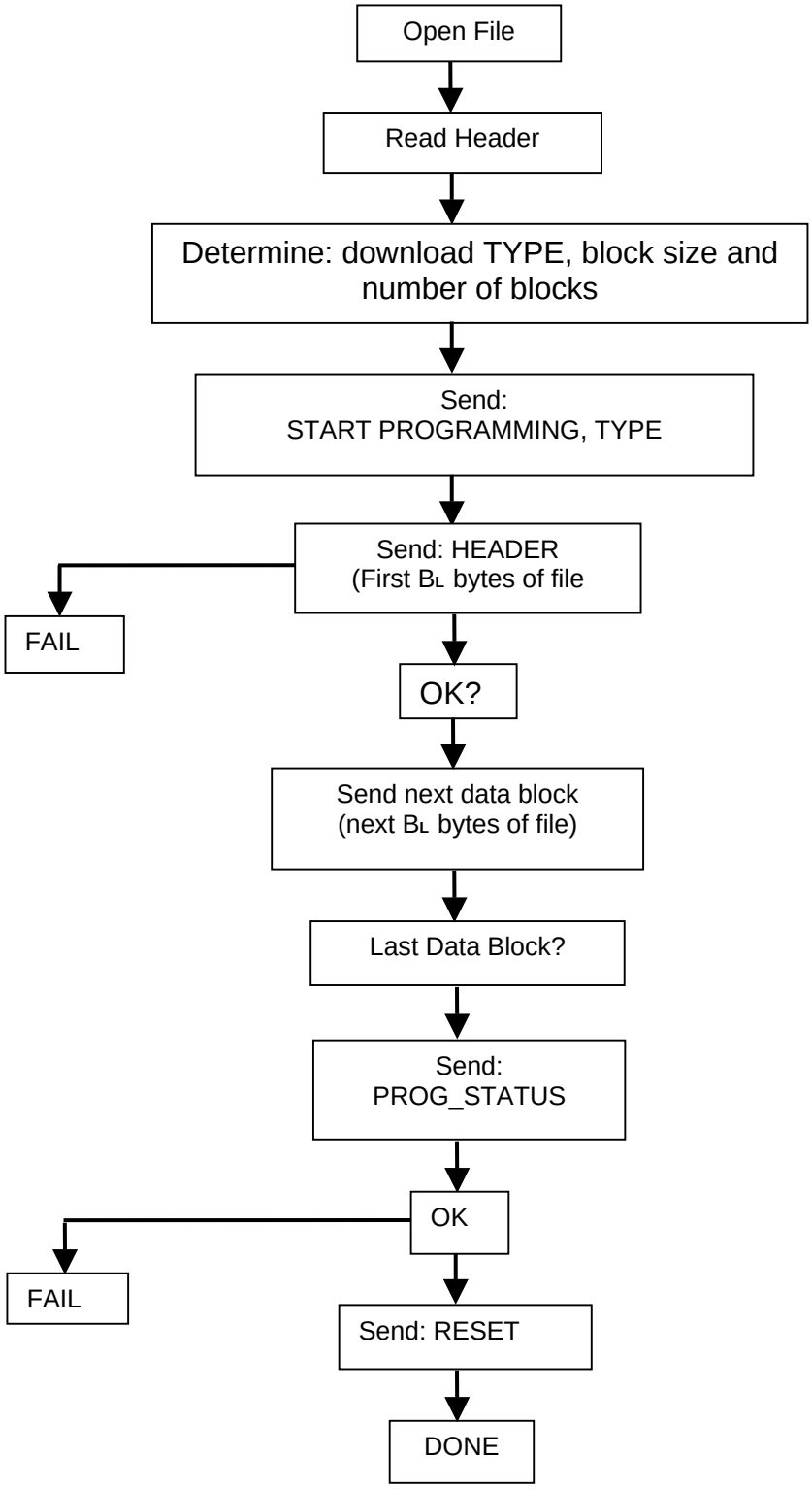

Figure 2 - Programming Flow Chart

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
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 (see table 14).

Data	Size/type	Notes
Unit Type	1 byte, integer	0x00 Note 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 bytes, integer	bytes 1 – 15 values
Security of Channel	15 bytes, integer	bytes 1 – 15 security
Real value multiplier	3 byte, integer	The value by which the channel values can be multiplied to show the true.
Protocol version	1 byte, integer	

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 values are just examples and would depend on the currency of the unit).
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.
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 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	Striming 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 (F2h) 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.

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
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 (Protocol version3)	0xE1, Channel No
Note cleared into cash box at reset (Protocol version 3)	0xE2, Channel No
Cash Box Removed (Protocol Version 3)	0xE3
Cash Box Replaced (Protocol Version 3)	0xE4
Bar Code Ticket Validated (Protocol Version 4)	0xE5
Bar Code Ticket Acknowledge (Protocol Version 4)	0xD1

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 it cycle.
Safe Jam: The slave has jammed and cannot return to service, the user cannot retrieve a note 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 note.
Disabled: The slave has been disabled, either by disabling all channels or the 5 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 > 0. This event is only reported if the Enable Protocol Version Events (0x19) command was sent in response to a SLAVE_RESET event.

Note cleared into cash box at reset: The validator has detected that a not was in the path at start up and has attempted to clear it into the cash box. If its channel was known, the channel No will be > 0. This event is only reported if the Enable Protocol Version Events (0x19) command was sent in response to a SLAVE_RESET event.
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 Enable Protocol Version Events (0x19) command was sent in response to a SLAVE_RESET event.
Cash Box Replaced: This event is only reported if the Enable Protocol Version Events (0x19) command was sent in response to a SLAVE_RESET event.
Bar Code Ticket Validated: The validator has detected and validated a TITO ticket and is held in escrow. The ticket details can be obtained by sending GET_BAR_CODE_DATA command. The ticket can be stacked on the next poll command or rejected with REJECT command.
Bar Code Ticket Acknowledge: The bar code ticket has reached is safe stack point (equivalent to note credit event).

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, Reteach 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

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 5 second poll time out has expired.
Fraud Attempt: The validator has detected an attempt to fish coins out of the unit.

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)
Get Device Setup	0x05
Payout Amount	0x33, value
Set Coin Amount	0x34, number, value
Get Coin amount	0x35, value
Halt Payout	0x38
Set Routing	0x3B, route, value
Get Routing	0x3C, value
Float	0x3D, min payout, float value
Get Minimum Payout	0x3E
Set Individual Coin Accept Inhibit	0x40, state, value
Empty	0x3F
Host Serial Number Request	0x14, Serial Nº

Table 21 – Smart Hopper Commands

Get Device Setup: 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 / type	Notes
Unit Type	1 byte, integer	0x03 Smart Hopper
Firmware Version	4 bytes, string	XX.XX (can include space)
Country Code	3 bytes, string	See Country Code Table. E.g. GBP
Protocol version	1 byte, integer	
Number of Coin Values (n)	1 byte, integer	
Coin Value 1	2 bytes, integer	e.g. 0x01, 0x00 = 0.01
.....	.....	Repeat for each coin value
Coin Value n	2 bytes, integer	

Table 22 - Response to Set-up request

Payout Amount: Five-byte command to set the value to payout, the first byte is the command. The following four bytes are the value to payout. Example: to pay out 2540 the command would be 33 EC 09 00 00. If the payout is possible the hopper 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

Set Coin Amount: Five-byte command, first byte is the command. The second and third bytes are the number of coins to add to the hopper. The fourth and fifth bytes are the value of each of the coins. Example to add 8 coins of value 0.10 the command would be 34 08 00 0A 00. If the number of coins sent is zero then the number of coins stored in the hopper for that value will be set to zero.

Get Coin Amount: Five-byte command that will return the coin counter for a given value in the hopper. The first byte is the command followed by four bytes representing the value of the coin. The response will be a two-byte count of the coins. Example: 37 0A 00 00 00 requests the number of 0010 value coins, the reply F0, 2C 01 indicates a count of 300 coins.

Set Routing: six-byte command, the first is the command. The second byte is the selected route, and the third to sixth is the value of the coin that the route should be applied to.

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

Float: Seven Byte command the first byte is the command. The second and third bytes are the value of the minimum payout, the final four bytes is the value to float to. Example: 3D 14 00 A8 61 00 00 will float to a value of £250 with a minimum possible payout of £0.20. 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 hopper	0x01
Cant float exact amount	0x02
Hopper Busy	0x03
Hopper Disabled	0x04

Set Individual Coin Accept Inhibit: 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 (0x00 Coin acceptance enabled, 0x01 Coin acceptance disabled). The final two bytes are the value of the coin to be inhibited or enabled. To enable the 2.00 coin the command would be 40, 01, C8, 00. Any values not supported by the Smart Hopper discrimination system will be inhibited automatically, regardless of this command state.

Empty: This command will route all detect coins to the cash box without reporting any value and reset all the stored coin counters to zero.

Host serial number request: This allows the host machine to send its serial number to the hopper. This may be used as an extra check that the host is authentic, however the fixed part of the encryption key provided this service.

6.9.3 SMART HOPPER RESPONSES TO POLLS

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
Emptying	0xC2
Emptied	0xC3
Fraud Attempt	0xE6, Fraud Code

Table 23 – Coin Hopper Responses to Polls

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: 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: 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 of 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.
Fraud Attempt: This will be reported if an attempt has been made to remove coins from the hopper.
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.

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)
Enable Payout Device	0x5C
Disable Payout Device	0x5B
Set Routing	0x3B, route, value
Get Routing	0x3C, value
Payout Amount	0x33,value
Get Note amount	0x35,value
Halt Payout	0x38
Float	0x3D, min payout, float value
Get Minimum Payout	0x3E
Empty	0x3F

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

Set Routing: Six-byte command, the first is the command. The second byte is the selected route, and the third to sixth are the value of the note that the route should be applied to. By default all note values are stacked.

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

Payout Amount: Five-byte command to set the value to payout, the first byte is the command. The following four bytes are the value to payout. Example: to pay out 5.00 the command would be 33 F4 01 00 00. If the payout is possible the Smart 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: Five-byte command that will return the note counter for a given value in the hopper. The first byte is the command followed by four bytes representing the value of the note. The response will be a two-byte count of the notes. Example: 35 E8 03 00 00 requests the number of 10.00 value notes (10 *100), the reply F0, 0A 00 indicates a count of 10 notes.

Float: Nine Byte command. The first byte is the command. The next four bytes are the value of the minimum payout, the final four bytes is the value to float to. Example: 3D F4 01 00 00 A8 61 00 00 will float to a value of £250 with a minimum possible payout of £5. If the float is possible the Smart Payout will reply with generic response OK. The unwanted notes will be stacked. 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: Single byte command, this will cause all notes to be sent to the stacker for removal.

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.

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

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.

APPENDIX A – COUNTRY CODES

Country	Abbr.	Country	Abbr.
Algeria	DZD	Japan	JPY
Arab (Un. Emir)	AED	Jordan	JOD
Argentina	ARA	Kenya	KES
Australia	AUD	Kuwait	KWD
Austria	ATS	Lebanon	LBP
Bahrain	BHD	Luxembourg	LUF
Belgium	BEF	Malaysia	MYR
Brazil	BRE	Mexico	MXP
Bulgaria	BGL	Morocco	MAD
Canada	CAD	Netherlands	NLG
China	CYN	Netherlands Ant.	ANG
C.I.S	RBL	New Zealand	NZD
Columbia	COP		
Cuba	CUP	Nigeria	NGN
Cyprus	CPY	Norway	NOK
Czechoslovakia	CSK	Oman	OMR
Denmark	DKK	Philippines	PHP
Egypt	EGP	Poland	PLZ
Euro	EUR	Portugal	PTE
Finland	FIM	Qatar	QAR
France	FRF	Rep. of Croatia	HRD
Germany	DEM	Rep. of Slovenia	SIT
Great Britain	GBP	Rumania	ROL
Greece	GRD	Saudi Arabia	SAR
Hong Kong	HKD	Singapore	SGD
Hungary	HUF	South Africa	ZAR
Iceland	ISK	Spain	ESP
India	INR	Sri Lanka	LKR
Indonesia	IDR	Sweden	SEK
Iran	IRR	Switzerland	CHF
Iraq	IQD	Syria	SYP
Ireland	IEP	Tunisia	TND
Israel	ILS	Turkey	TRL
Italy	ITL	United States	USD

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

```
/*
|-------------------------------------
|      c INNOVATIVE TECHNOLOGY LTD 2000
|-------------------------------------|
| TITLE  :CRC Functions       |Language : C
| DRG No :                    |Project  :
| Author  : P Dunlop          |Date     :17-04-2000
| Revision: A
|-------------------------------------|
| Issue No.  |    Mod No.  |      DATE  |     Mod By.
|-------------------------------------|
|   ISSUE A  |             |            |
|-------------------------------------|
|External files Used:
|   Includes:
|   Linked:
|   Config:
|   Docs:
|-------------------------------------|
|  CRC FUNCTIONS FOR POLYNOMIAL (X^16)+(X^15)+(X^2)+1 AS USED IN SSP    |
|  CRC IS INITIALISED WITH THE SEED 0xFFFF                              |
|-------------------------------------
*/
#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;
}
```