Innovative Technology Limited ®

SSP Smiley® Secure Protocol

Document Issue 12 - Protocol Version 3

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 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
Change 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

Table Of Contents

1.0 Introduction

This manual describes the operation of the Smiley® Secure Protocol - SSP as fitted with Firmware Version 1.10 or greater.

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

Innovative Technology Ltd.
Derker Street
Oldham
England
OL1 4EQ
Tel: +44 (0) 161 626 9999
Fax: +44 (0) 161 620 2090
Email support@innovative-technology.co.uk
web site www.innovative-technology.co.uk

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. To provide extra security the protocol can operate in an encrypted mode to protect the system from fraud through bus monitoring.

To provide this security a constantly changing random 64 bit key is used. 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
- 64 Bit 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. (Currently not implemented)

5.1 Packet Format.

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).

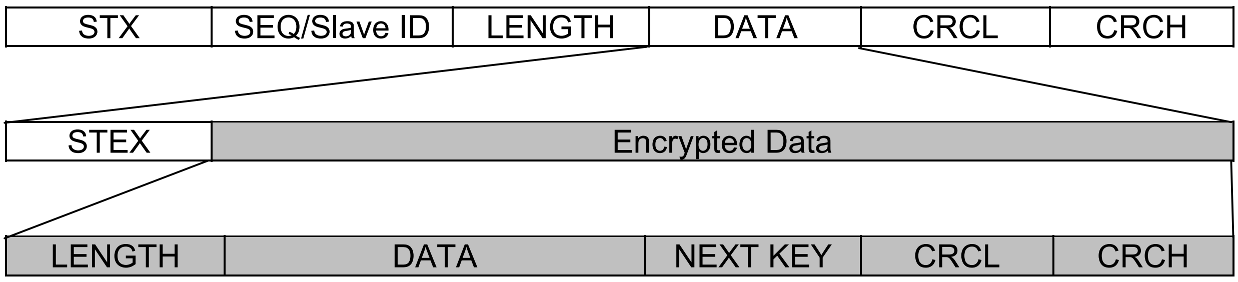

Figure 1 – Encrypted Data Format

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, the next key or the CRC.
DATA:	Commands or data to be transferred.
NEXT KEY:	The key needed to decrypt the next encrypted packet from the host.
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.

5.2 Encryption Keys.

At power up the first encrypted data packet sent to the peripheral will be encrypted using the peripherals serial number (the host requests and stores the serial number of each peripheral when it is installed – see control layer).

After the first transmission data is encrypted/decrypted using the key contained in the last packet, the host generates this randomly. The peripheral will always reply to an encrypted packet with data encrypted using the same key as the original packet from the host.

After the data has been decrypted the CRC algorithm is preformed on all bytes including DATA, NEXT KEY, CRCL and CRCH. 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.

5.3 Encryption Algorithm.

The encryption algorithm has a 64-bit key. This is only a short key but when combined with the mechanism for changing the key in every packet the system provides a high level of security. Appendix B contains C source code for encryption and decryption.

The algorithm will easily translate into assembly code as long as the XOR is an operation. The routines works on blocks of 16 bytes that are packed into an array of four long integers, the key is 8 bytes long and is packed into an array of 2 long integers. If the data to be sent is not a multiple of 16 bytes then the remaining bytes are packed out with zeros (see tables 3).

5.4 Encryption Example.

Convert key into long integer

Key()	0	1	2	3	4	5	6	7
Value	Ox67	Ox45	Ox23	Ox01	OxEF	OxCD	OxAB	Ox89
L_key()	0				1			
Value	Ox01234567				Ox89ABCDEF			

Get data bytes

Data	0	1	2	3	4	5	6	7	8	9	10
Value	OxFF	OxEE	OxDD	OxCC	OxBB	OxAA	Ox99	Ox88	Ox77	Ox66	Ox55

Convert data into array of 4 long integers any unused bytes set to Ox00

Data	0	1	2	3
Value	OxCCDDEEFF	Ox8899AABB	Ox00556677	Ox00000000

Pass data through encryption algorithm.

E_Data	0	1	2	3
Value	Ox1500F4F6	OxF56E7CBA	OxDA441723	Ox5D2743D2

Break up into individual bytes for transmission:

E_Data	0				1				2				3			
Value	Ox1500F4F6				OxF56E7CBA				OxDA441723				Ox5D2743D2			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Value	F6	F4	00	15	BA	7C	6E	F5	23	17	44	DA	D2	43	27	5D

Tables 3 – Encryption Examples

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	0xFO
Command not known	0xF2
Wrong number of parameters	0xF3
Parameter out of range	0xF4
Command cannot be processed	0xF5
Software Error	0xF6
FAIL	0xF8

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.

6.4.3 Remote Programming (Version 2.65 and later).

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.

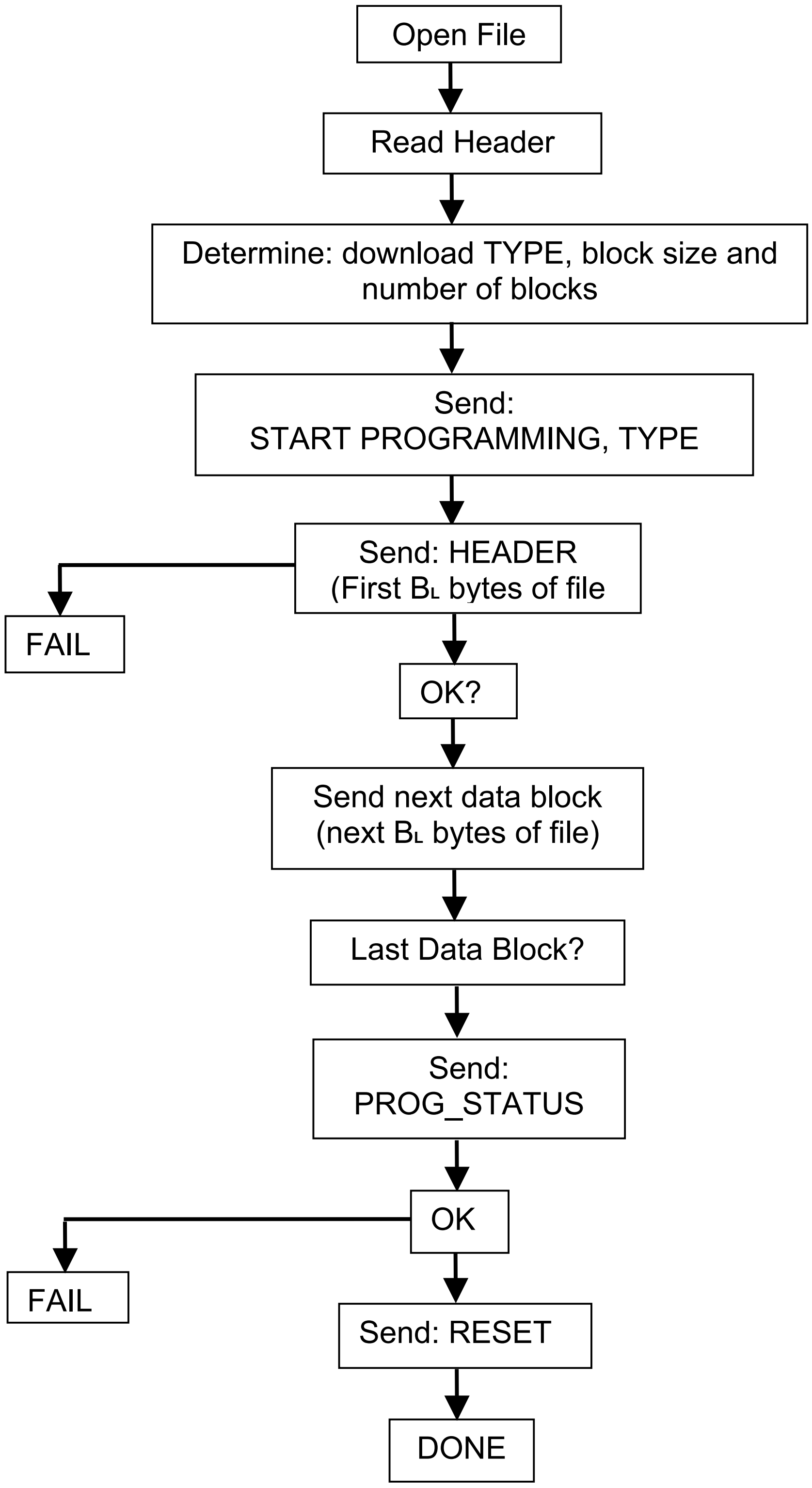
Open File
Read Header
Determine: download TYPE, block size and number of blocks
Send: START PROGRAMMING, TYPE
Send: HEADER (First BL bytes of file
FAIL
OK?
Send next data block (next BL bytes of file)
Last Data Block?
Send: PROG_STATUS
OK
FAIL
Send: RESET
DONE

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

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
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 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.
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 15). Specifics of note validation not shown to protect integrity of manufacturers security (Version 2.66 and later).

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

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.

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

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.
Fraud Attempt: The validator has detected an attempt to fish notes out of the Stacker.

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, 0x24 – Update route for channel 2 to 00100100 (routes 3 and 6).

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
Credit, n	0xEE, CHANNEL N^o
Rejected	0xEC
Coin Routed	0xE5, route N^o
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.
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 Coin Hopper

6.7.1 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 Coin Hopper Commands

Action	Command code (HEX)
Dispense	0x13, N° of coins
Host Serial Number Request	0x14, Serial N°
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 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.

6.8.1 Basic Card Reader Operation.

The basic card reader will allow a small amount of data to be securely stored on a removable card.

The reader / card combination provides the following features:

- Each card reports a unique 32-bit ID number.
- A small amount of data can be stored on each card (CS1 16 bytes).
- The host must provide a 32-bit password to access the card.
- The card provides a 24-bit password to the host.
- The host can display messages on the readers display.

Before the card will reports its ID number or allow access to the data on the card the card must be authorised.

The process to do this involves the following two steps:

Firstly the host must provide the reader with the correct password for the card, the card will then provide its ID number and a system password.
The host must check that the password provided by the card is correct for this system.

Once this process is complete then the host can read or write data from the card. The states of the card reader and transitions between states are shown below (see figure 3).

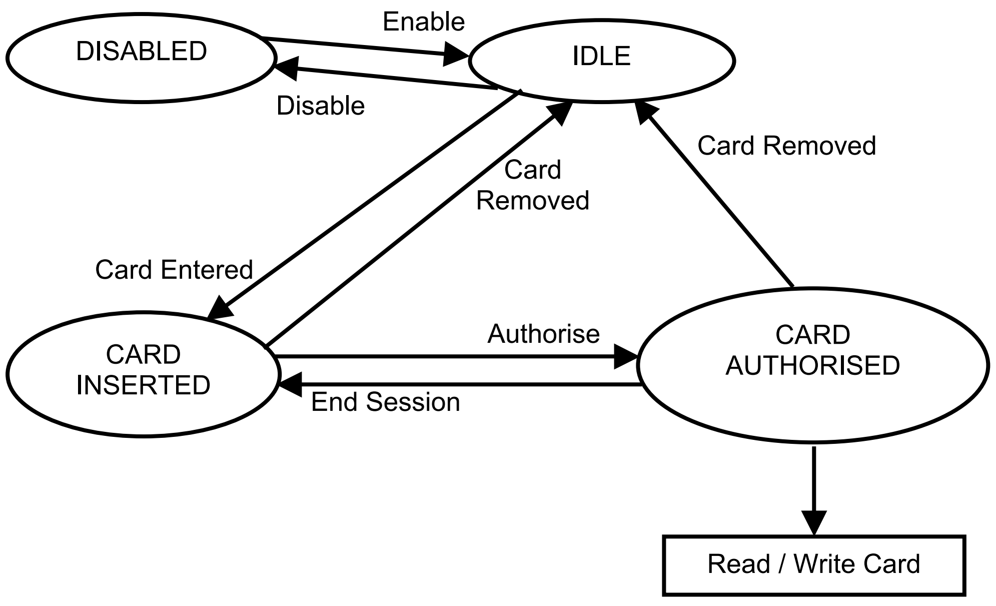

Figure 3 Card Transition States

6.8.2 Basic Card Reader Commands.

Action	Command code (HEX)
Authorise Card	0x20, Card Password
Read Card	0x21
Write Card	0x22, Data
End Session	0x23
Display Message	0x24, Message, 0x00
Set-up Request	0x15

Table 24 - Basic Card Reader Commands

Authorise Card: Used to authorise a card, this must be successfully completed before the card can be read from or written to. The command should be followed by a four-byte card password. If the card accepts the password then the reader will respond with OK, a four byte card ID and a three-byte system password, otherwise it will respond with FAIL. (See table 25) If the returned system password is acceptable to the host then the card is accepted, if not then the host should use the END SESSION command to reject the card.

Response	Code
Password Accepted – OK	0xF0,Card ID (4), System Password (3)
Password Rejected – FAIL	0xF8

Table 25 - Password Commands

Read Card: This command returns all the data from the card; the Storage Capacity field of the Set-up Request response determines the number of bytes returned. This command can only be used after a successful Authorise Card sequence.
Write Card: This command is used to write data to the card, the data to be written to the card should follow the command. The reader will respond with a generic response; OK if the data was written successfully and FAIL if there was a problem. This command can only be used after a successful Authorise Card sequence.
End Session: This command is used to end the session with the card after reading and writing to the card, or if the card ID or system password are unacceptable.
Display Message: The host can use this command to display messages on the reader display. The command should be followed by a NULL terminated string to be displayed. The message will be displayed once only. If a previous message is still being displayed then the new message will be displayed once the previous one has finished. The maximum message length is 240 characters.
Set-up Request: Single byte command, used to request information about a slave. Slave will return the following data: Unit Type, Firmware version, Storage Capacity, Unit Capability, and Version of Protocol (see table 26).

Data	Size/type	Notes
Unit Type	1 byte, integer	0x03 - Basic Card Reader
Firmware Version	4 bytes, string	XX.XX (can include space)
Storage Capacity	1 byte, integer	Number of bytes (CS1 – 16)
Units Capability	1 byte, integer	0x00 Read only, 0x01 W/R
Protocol version	1 byte, integer	

Table 26 Response to Set-up request

6.8.3 Basic Card Reader Responses to Polls.

Event / State	Event Code
Card Inserted	0xC0
Card Authorised	0xC1, Password
Card Removed	0xC2
Disabled	0xE8
Unit Fault	0xC3, Fault Code

Table 27 - Basic Card Reader Responses to Polls

Card Inserted: This state is entered when a card is entered into the reader; it can only be entered from the idle state. The state is left when the card is authorised or when the card is removed.
Card Authorised: This state is entered after a successful card authorise sequence. This is the only state in which the card can be read from and written to. The state is exited after the host issues an End Session command (returning to the Card Inserted state), or if the card is removed (returning to the idle state after reporting a Card Removed event).
Card Removed: This event is reported when the card is removed from the reader, the reader will return to the idle state.
Disabled: This state is the initial state after power up; it is also entered after the host issues a Disable command. The display is off, and all card entered are ignored.
Unit Fault: In the event that of the card reader reports a Unit Fault the event code 0XC3 will be transmitted to the host machine.

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 Routines.

B.1 Encryption routine

```
/*********************************************************************
   Function CODE
Function to encrypt 16 bytes of data (4 long ints)
Usage:   :      code (long_data, long_current_key);
Parameters:     data()  - 16 bytes to be encrypted packed into 4 long ints
                Key()  - key to use packed into array of 2 long ints
Returns:        data() - 16 bytes of encrypted data packed into 4 long ints
Locals: w,x,y,z,delta,n
/*********************************************************************
void code(long* data, long* key)  {
        unsigned long w=data[0],x=data[1],y=data[2],z=data[3], sum=0,   /* set up */
        delta=0x9e3779b9, n=32 ;            /* a key schedule constant */
        while (n-->0) {                  /* basic cycle start */
                sum += delta ;
                w += (z<<4)+key[0] ^ z+sum ^ (z>>5)+key[1] ;
                x += (w<<4)+key[1] ^ w+sum ^ (w>>5)+key[0] ;
                y += (x<<4)+key[0] ^ x+sum ^ (x>>5)+key[1] ;
                z += (y<<4)+key[1] ^ y+sum ^ (y>>5)+key[0] ;
        }                                          /* end cycle */
        data0]=w ; data[1]=x ; data[2]=y ;data[3]=z;
}
```

B.2 Decryption routine

```
/*********************************************************************
   Function DECODE
Function to encrypt 16 bytes of data (4 long ints)
Usage:   :      decode (long_data, long_current_key);
Parameters:     data() - 16 bytes of encrypted data packed into 4 long ints
                Key()  - key to use packed into array of 2 long ints
Returns:        data()  - 16 bytes of decrypted data packed into 4 long ints
Locals:         w,x,y,z,delta,n
/*********************************************************************
void decode(long* data, long* key)  {
        unsigned long n=32, sum,w=data[0],x=data[1], y=data[2], z=data[3],
        delta=0x9e3779b9 ;
        sum=delta<<5 ;
        while (n-->0) {            /* start cycle */
                z-= (y<<4)+key[1] ^ y+sum ^ (y>>5)+key[0] ;
                y-= (x<<4)+key[0] ^ x+sum ^ (x>>5)+key[1] ;
                x-= (w<<4)+key[1] ^ w+sum ^ (w>>5)+key[0] ;
                w-= (z<<4)+key[0] ^ z+sum ^ (z>>5)+key[1] ;
                sum-=delta ;
        }                                     /* end cycle */
        data[0]=w ; data[1]=x ; data[2]=y ;data[3]=z;
}
```

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;
}
```