RETAIL EBDS PROTOCOL SPECIFICATION

(with *M/POST for EBDS*)

TITLE	RETAIL EBDS PROTOCOL SPECIFICATION (with M/POST for EBDS)
NUMBER	20105-002850131-PS
ISSUE	G2
PCN	500000008441
DATE	August 11, 2008
AUTHORS	Peter Camilleri

Change History

Issue	Date	Description	Author(s)
N/A	March 2, 2006	Initial Draft	Peter Camilleri
N/A	June 26, 2007	Continuing Work In Progress	Peter Camilleri
N/A	July 9, 2007	Continuing Work In Progress	Peter Camilleri
N/A	July 13, 2007	Added chapter for EBDS section of the MEI Point of Service Toolkit (M/POST).	Peter Camilleri
N/A	July 24, 2007	First complete draft.	Peter Camilleri
N/A	July 27, 2007	Added sections for bindings for ActiveX, LINUX, .NET and JAVA to be filled in at a later time. Added Quick reference and Hex/Binary Conversion tables	Peter Camilleri
N/A	Aug 23, 2007	Updated the introduction to cover M/POST. Added warning about the loopback failure mode.	Peter Camilleri
N/A	Sept 10, 2007	Added some brief notes on .NET bindings and the recommended handling of events.	Peter Camilleri
N/A	Sept 19, 2007	Updated QueryDeviceCapabilities command. Added QueryBNFStatus, QueryAcceptorApplicationID, and QueryAcceptorVariantID commands. Also added CapBNFStatus, BNFStatus, CapApplicationID, ApplicationID, CapVariantID, VariantID and CapTestDoc properties to M/POST	Peter Camilleri
N/A	Oct 31, 2007	Added the DebugLogPath property and a section for the M/POST demo program.	Peter Camilleri
N/A	Nov 12, 2007	Added VB.NET Event handling section. Added an errata entry to the BillTypeEnables property details.	Peter Camilleri
N/A	Nov 16, 2007	Corrected the model id values for Zt1200 US and the entries for the bill tables associated with those tables.	Peter Camilleri
N/A	Nov 28, 2007	Incorporated Dave McLaughlin's comments	Peter Camilleri
N/A	Jan 7, 2008	Added the OLE bindings section	Peter Camilleri
N/A	Jan 9, 2008	Added table clarifying message type 7x usage.	Peter Camilleri
N/A	Jan 10, 2008	Added an ASCII conversion table.	Peter Camilleri
N/A	Jan 22, 2008	Added the Set Bezel command.	Peter Camilleri
N/A	Jan 29, 2008	Updated the Set Bezel command. Updated descriptions of the M/POST demo.	Peter Camilleri
N/A	Feb 22, 2008	Clarified the intent of unused (0) bits in commands and replies. Updated the SetExpandedNoteInhibits command to indicate the two possible replies.	Peter Camilleri
N/A	Feb 25, 2008	Documented the alternate sequence for determining Device Capabilities.	Peter Camilleri
N/A	April 4, 2008	Updated reply time-outs to allow for USB links.	Peter Camilleri
N/A	April 18, 2008	Added preliminary Linux bindings information.	Peter Camilleri
N/A	April 23, 2008	Added PID and VID settings for USB connections.	Peter Camilleri
G1	May 13, 2008	First release.	Peter Camilleri
G2	August 11, 2008	Updated with data for the Bunch Note Feeder	Peter Camilleri

Table of Contents

1. Introduction 7

1.1 Scope 7

1.2 Recommendation/Warning Icons 8

1.3 Product Icons 8

1.4 Varieties of EBDS 9

2. A Protocol Overview 10

3. The Physical Layer 11

3.1 RS-232 11

3.2 RS-485 11

3.3 Optically Isolated 11

3.4 USB 11

3.4.1. USB Device Identifiers 12

4. The Data Link Layer 13

4.1 Data Packet Construction 13

4.1.1 More about STX/ETX and the CHK 13

4.2 Normal (Polled) Mode 13

4.2.1 Normal Mode Timing 14

4.3 Special (Interrupt) Mode 14

4.3.1 Special Mode Timing 15

4.3.2 ABDS and Special (Interrupt) Mode 15

4.3.3 Flash Download and Special (Interrupt) Mode 15

4.4 Receiving a reply 16

4.4.1 Loopback Error Handling 17

4.5 ACK/NAK Processing 17

4.5.1 ACK/NAK Examples: 18

4.5.2 ACK/NAK Timing Requirements 19

5. The Network Layer 20

5.1 Device Type Message Routing 20

5.2 ABDS Message Routing 21

5.2.1 ABDS Timing Requirements 22

5.2.2 ABDS and Special (Interrupt) Mode 22

6. The Session Layer 23

6.1 Variations in Power Up reporting 23

6.2 Starting in Download mode 23

6.2 Ending the Session 24

7. The Presentation Layer (for Bill Acceptor) 25

7.1 The Omnibus Command 25

7.1.1 Omnibus Command 26
7.1.2 Standard Omnibus Reply 29
7.1.3 Extended Omnibus Bar Code Reply 33
7.1.4 Extended Omnibus Expanded Note Reply 33
7.1.5 Extended Omnibus Expanded Coupon Reply 36

7.2 The Calibrate Command 37

7.3 The Download Firmware Command 38
7.3.1 The pace of communications during download. 38
7.3.2 Overview of the Flash Download Process 38
7.3.3 Download Flow Diagram 40

7.4 The Auxiliary Commands 42
7.4.1 Query Software CRC 42
7.4.2 Query Cash Box Total 43
7.4.3 Query Device Resets 44
7.4.4 Clear Cash Box Total 45
7.4.5 Query Acceptor Type 45
7.4.6 Query Acceptor Serial Number 46
7.4.7 Query Acceptor Boot Part Number 48
7.4.8 Query Acceptor Application Part Number 48
7.4.9 Query Acceptor Variant Name 49
7.4.10 Query Acceptor Variant Part Number 50
7.4.11 Query Acceptor Audit Life Time Totals 51
7.4.12 Query Acceptor Audit QP Measures 53
7.4.13 Query Acceptor Audit Performance Measures 54
7.4.14 Query Device Capabilities 56
7.4.15 Query Acceptor Application ID 57
7.4.16 Query Acceptor Variant ID 58
7.4.17 Query BNF Status 59
7.4.18 Set Bezel 59
7.4.19 Acceptor Soft Reset 60

7.5 The Extended Commands 61
7.5.1 Query Expanded Note Specification 61
7.5.2 Set Expanded Note Inhibits 63
7.5.3 Set Escrow Timeout 64
7.5.4 Set Asset Number 65
7.5.5 Query Value Table 65
7.5.6 Set Extended PUP Mode 66

7.6 Processing States 67
7.6.1 Processing States in Non-escrow Mode 68
7.6.2 Processing States in Escrow Mode 69

8. The Application Layer (for Bill Acceptor) 70

8.1 Application Startup Tasks 70

8.2 Application Currency Handling 71
8.2.1 Handling Money in Terse Mode: 71
8.2.2 Handling Money in Expanded Mode: 71
8.2.3 Recommended money handling flowchart: 72
8.2.4 Controlling the orientation of accepted bills: 73
8.2.5 Improved control of the orientation in expanded mode: 74

8.3 Determining the Firmware Version 74
8.3.1 Firmware Version in Classic EBDS: 74

8.3.2 Firmware Version in Extended EBDS: 75

8.4 Handling Acceptor Exceptions 75
8.4.1 Bill Acceptor does not respond to a poll: 75
8.4.2 Bill Acceptor does not respond for an extended period: 75
8.4.3 Bill Acceptor Status: Cheated 75
8.4.4 Bill Acceptor Status: Rejected 76
8.4.5 Bill Acceptor Status: Jammed 76
8.4.6 Bill Acceptor Status: Stacker Full 76
8.4.7 Bill Acceptor Status: Cashbox Removed 76
8.4.8 Bill Acceptor Status: Paused 76
8.4.9 Bill Acceptor Status: Calibration in Progress 76
8.4.10 Bill Acceptor Status: Power Up 77
8.4.11 Bill Acceptor Status: Invalid Command 77
8.4.12 Bill Acceptor Status: Failure 77
8.4.13 Bill Acceptor Status: Stalled 77
8.4.14 Bill Acceptor Status: Flash Download 77

9. MEI Point of Service Toolkit (M/POST) 78

9.1 M/POST for EBDS Overview: 78
9.1.1 M/POST for EBDS **Acceptor** Properties: 78
9.1.2 M/POST for EBDS **Bill** Properties: 80
9.1.3 M/POST for EBDS **Coupon** Properties: 80
9.1.4 M/POST for EBDS **DocType** Enumeration: 80
9.1.5 M/POST for EBDS **Orientation** Enumeration: 81
9.1.6 M/POST for EBDS **OrientationCtl** Enumeration: 81
9.1.7 M/POST for EBDS **PowerUp** Enumeration: 81
9.1.8 M/POST for EBDS **PupExt** Enumeration: 81
9.1.9 M/POST for EBDS **State** Enumeration: 81
9.1.10 M/POST for EBDS **BNFStatus** Enumeration: 81
9.1.11 M/POST for EBDS **Bezel** Enumeration: 82
9.1.12 M/POST for EBDS **Acceptor** Methods: 82
9.1.13 M/POST for EBDS **Acceptor** Events: 82
9.1.14 M/POST for EBDS **Acceptor** Properties Details: 83
9.1.15 M/POST for EBDS **Acceptor** Methods Details: 94
9.1.16 M/POST for EBDS **Acceptor** Events Details: 97

10. M/POST Bindings for ActiveX 101

10.1 Connecting to the M/POST DLL 101

10.2 Handling Events in Visual Basic 6 101

10.3 Differences from the M/POST model 102

10.4 A note on array index values. 103

10.5 A note on Boolean property values. 103

11. M/POST Bindings for .NET 104

11.1 Connecting to the M/POST DLL 104

11.2 Handling M/POST Events in C# 105

11.3 Handling M/POST Events in VB.NET 106

12. M/POST Bindings for Linux 107

12.1. Using the MPOST_Linux Library 107

12.2. Handling M/POST Events in Linux 107

12.3. MPOST Linux Demo Program 108

13. M/POST Bindings for JAVA 109

14. The M/POST Demo Program 110

14.1 The Launcher 110

14.2 The Control Panel 111

14.2.1 The Main Tab 112
14.2.2 The Capabilities Tab 115
14.2.3 The Properties Tab 115
14.2.4 The Bill Set Tab 118
14.2.5 The Bill Values Tab 119
14.2.6 The Device Info Tab 120

15. Device Harness and Connection 121

15.1 EBDS Harness Options for the Series 2000 Bill Acceptor: 121

14.1.1 Custom Harness Design for the Series 2000 Bill Acceptor: 121

15.2 EBDS Harness Options for the Cashflow-SC Bill Acceptor: 122

14.2.1 RS-232 Harness Configuration: 122
15.2.2 USB Harness Configuration: 123

15.3 Legacy, Series (ZT) 1000 Harness Options 124

15.4 Legacy, Series 3000 Harness Options 124

16. Quick Reference 127

17. Hex/Binary and ASCII Data Conversion 128

1. Introduction

1.1 Scope

The scope of this document is to specify the EBDS protocol with the goal of making it easier to incorporate EBDS devices into business solutions. To this end, the intended audiences of this document are host system software developers, technical support specialists and system integrators.

Firstly, to serve this group better, this document employs a structured approach based on the OSI networking model. This model has proven to be a solid base for the implementation of device to device communications of all sorts.

Secondly, this document will focus on the needs of developers in the retail, kiosk, self-check-out and cash-drop markets.

Thirdly, this document serves to document the MEI / Point Of Service Toolkit (M/POST). This tool supports EBDS and greatly reduces the effort required to implement an EBDS host. Users of M/POST may wish to skip directly to section 9 and refer to earlier sections as needed.

Finally, this document will examine EBDS as used in the following Retail products:

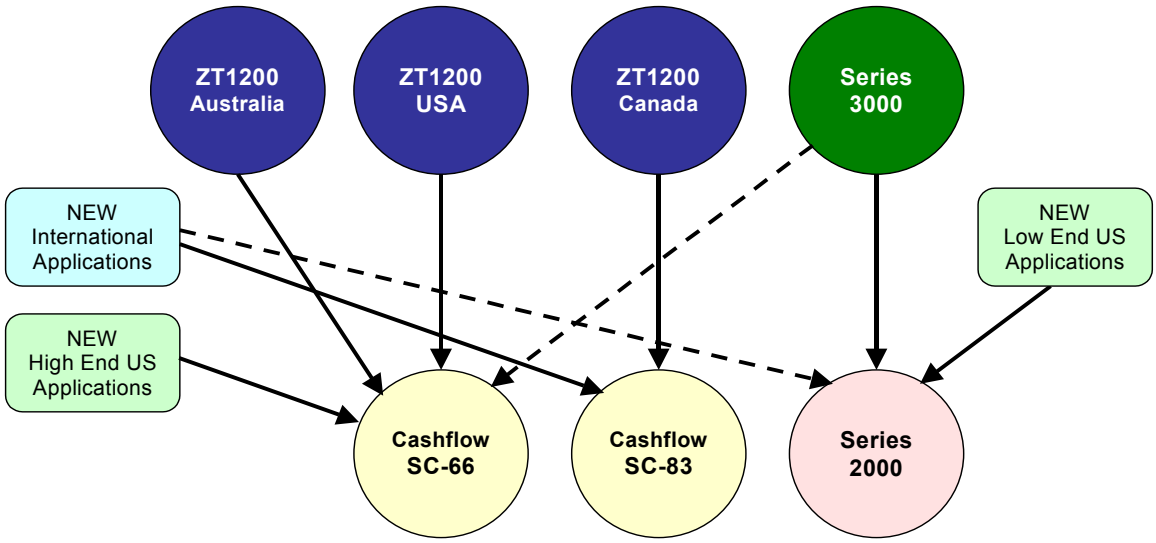
Family	Model	Variants
Series 2000	AE2600	US, Australia, Canada,
	AE2800	US, Argentina, Brazil
Cashflow SC	SC-66	Flexible variants
	SC-83	Flexible variants
	SC-85	Special UK product

The following legacy products are also discussed to aid in the host code migration process:

Family	Model	Variants	Description
Series 1000	ZT1200	US, Australia, Canada	Discontinued products
Series 2000	AE2600	US	Earlier model hardware.
Series 3000	LE, RS, EX	US, Australia	Discontinued products
Cashflow SC	SC-66	US, Australia	Earlier version software.

In all cases, MEI advises that the best operation of cash devices is obtained when they are loaded with the most current firmware for that product.

The recommended migration paths from older products to newer is illustrated below:



1.2 Recommendation/Warning Icons

The following warning icons will be used to highlight best practices and potential problem areas.

Recommended	A feature that is recommended as a "best" practice.
Non-Retail	A feature that is not suitable for retail applications.
Incompatible	Features that are incompatible or mutually exclusive.
Deprecated	A feature that is no longer recommended.
Obsolete	A feature that used to be supported but is no longer. In general, any features or commands so marked should be avoided.

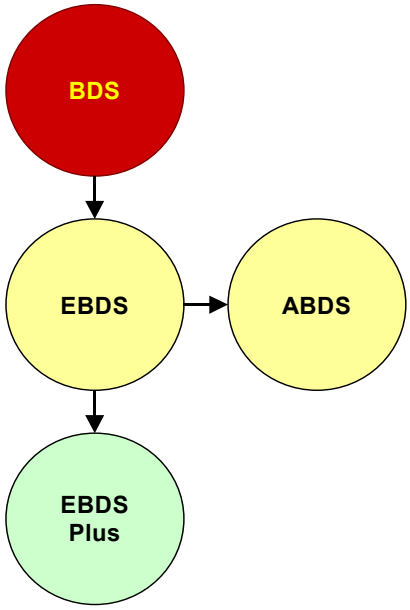
1.3 Product Icons

Some features of EBDS are specific to or vary by the product implementing them. To indicate this, the following icons are used:

S1K	All series 1000 retail products (ZT120x)
S2K	All series 2000 retail products (AE2600, AE2800)
Gen2D	Series 2000, AE2800 and newer AE2600 US products.
Gen2B/C	Older Series 2000 AE2600 US products.
S3K	All series 3000 products (LE/RS/EX).
CFSC	All Cashflow-SC retail products.
BNF	A Cashflow-SC with the Bunch Note Feeder Option
CFMC	Cashflow SC-66 retail products with older software.

1.4 Varieties of EBDS

The EBDS protocol has evolved over time. As it has changed, the naming of the versions has also changed. The family tree of EBDS is illustrated below.



Acronym	Meaning
BDS	**Bi**D**i**rectional **S**erial Protocol **Obsolete**
EBDS	**E**nhanced **Bi**D**i**rectional **S**erial Protocol
ABDS	**A**ddressable **Bi**D**i**rectional **S**erial Protocol
EBDS Plus	**E**nhanced **Bi**D**i**rectional **S**erial Protocol **Plus**

- The BDS protocol is no longer supported on any current cash handling products. This protocol is discussed here only to extent needed to point out migration issues.
- The EBDS protocol is widely used on Series 2000 as well as legacy Series 1000, 3000 and older SC-66 models.
- The ABDS protocol is supported by Series 3000 and the Cashflow-SC with the RS-485 option installed. This protocol allows multiple cash acceptors to be connected to a single, multi-drop, host port.
- EBDS Plus is similar to EBDS with the additional option of extended note reporting and several command-oriented extensions to the original omnibus command packet.

2. A Protocol Overview

The ISO OSI Network Model is used to describe networks of all sorts and a Computer connected to an EBDS device is no exception. This is illustrated in the following layered breakdown of a connection to such a device.

Layer	Host Computer	Units of Transmission	EBDS Device
7	Application	Actions/Events	Application
6	Presentation	Commands/Responses	Presentation
5	Session	Messages	Session
4	Transport	Messages	Transport
3	Network	Packets	Network
2	Data Link	Frames	Data Link
1	Physical	Bits	Physical
	Interconnecting Medium		

The purpose of these layers is explained below:

Layer	Description
Application	This is the high level software on the computer and the functionality of the device.
Presentation	The command/response layer that interacts with the application.
Session	This level deals with multiple interactions between the computer and the device. In EBDS this corresponds to power up/down and application start/finish.
Transport	The break up and recombination of larger transactions into smaller packets. In most cases, EBDS transactions fit inside a single packet.
Network	Network address and routing used to guide the packet to the correct destination
Data Link	The collection of many bits into a meaning unit or packet, including error detection.
Physical	The voltage/current and timing for a single bit in a frame.
Interconnecting Medium	The connectors and wires/transmission medium of the connection.

3. The Physical Layer

Data in EBDS is transmitted using asynchronous data bytes. These may be formatted as:

9600bps, 7 data bits, even parity, one stop bit
1200bps, 7 data bits, even parity, one stop bit **Obsolete**

The EBDS protocols support a number of different types of physical layers. These are laid out below:

3.1 RS-232

The RS-232 interface is supported across all products. It should be noted though that some products require the use of an adapter harness to be used in this mode. EBDS uses the minimum, three wire, configuration (GND, TXD, RXD) with no handshaking lines. The data transmission across the line corresponds to the EIA RS-232 spec, which is not repeated here. The formal specification may be found in ITU-T Recommendation (formally CCITT Recommendation) V.24. A more useful, if less formal specification may be found at the Wikipedia web site under the topic RS-232.

3.2 RS-485

The RS-485 interface is a specialized interface used to support the multi-drop features of the ABDS protocol. It is supported by the RS-485 interface adapter harness of the Series 3000 and by the Cashflow-SC with the RS-485 Interface Board option. When RS-485 is utilized, all communications occur over a single balanced pair data line in a true half-duplex mode. No handshaking lines are utilized.

3.3 Optically Isolated

Non-Retail This interface type utilizes optically isolated drivers. To maintain isolation, the ground reference and power required by the host side of the connection must all be supplied by the host. The optically isolated interface is not normally used in retail applications. However, this interface may find application in cases where the host system is physically remote from the device in an electronically noisy environment.

3.4 USB

The EBDS protocol may also be embedded inside the USB virtual communications port protocol. This may be accomplished with an RS-232 Virtual Com Port Cable or through the use of the appropriate harness (for Series 2000) or interface board (for Cashflow-SC). No handshake lines are utilized.

The protocol layers involved in embedding EBDS in USB are covered in publicly available USB specs and are not covered in this document.

Incompatible Some USB Virtual Com Cables do not support the data transfer parameters required by EBDS (9600, 7, E, 1). These cables will not work with an EBDS device. In addition, some of these cables introduce delays that may disrupt some EBDS operations.

3.4.1. USB Device Identifiers

Normally, MEI devices that utilize a USB connection are supplied with an installation program that sets up the required devices drivers and settings. Under embedded systems or some operating systems, this is not the case and the systems integrator will need to configure the drivers manually.

To assist in manual configuration, the following table sets out the crucial USB parameters for various MEI products:

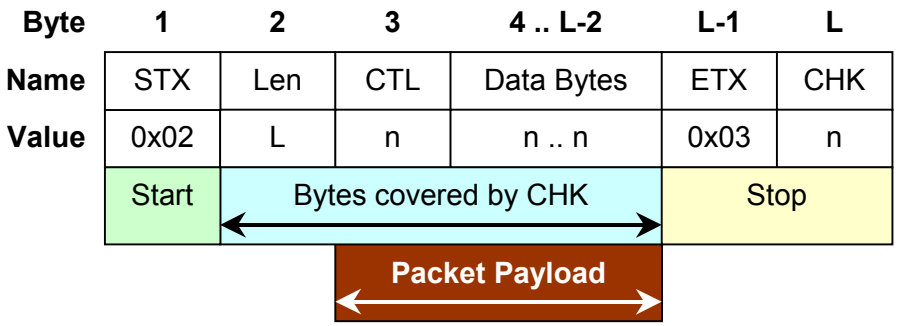
Device	Vendor ID for MEI (VID)	Product ID for device (PID)
Cashflow-SC or Cashflow-SCL, models 6628 or 8328.	0x0BED	0x1100
Series 2000 models AE2800 or AE2600 with optional USB harness kit.		0x1101

For further information on drivers, please see the Silicon Laboratories web site at: http://www.silabs.com/ and look up virtual communications drivers for the CP2102. For WinCE® hosts, a custom integration with the USBXpress® development kit will be required.

4. The Data Link Layer

4.1 Data Packet Construction

Data packets, whether from the host or the device, are laid out according to the following plan:



Where:

STX	The ASCII STX character. Hexadecimal value 0x02. The start of a packet.
Len	The count of all of the bytes in the packet.
CTL Byte	A mix of the ACK/NAK field, a reserved area, and the command area.
Data Bytes	The data bytes of the packet. This is Len-5 bytes worth.
ETX	The ASCII ETX character. Hexadecimal value 0x03. The end of a packet.
Checksum	The XOR checksum of bytes 2 through Len-2.

4.1.1 More about STX/ETX and the CHK

It is very important to realize that the values used for STX and ETX, namely 0x02 and 0x03 are NOT unique within the packet structure. It is possible (if unlikely) for any of the data bytes or the check byte to be of these values. Thus the host code must not be written to always start a packet when a 0x02 is encountered or end one when a 0x03 in found. Instead, the length byte should be used to determine when a packet ends and the timeout when it is lost.

The data link layer can be operated in one of two modes, Normal Mode or Special Mode.

4.2 Normal (Polled) Mode

In normal mode, EBDS operates as a standard host/peripheral system with all transactions initiated by the host and only replies generated by the peripheral. All communications take place in packets that follow the rules in section 3.1.

In polled mode, the host system is responsible for running transactions to accomplish three goals:

1) Sending any commands the host needs to set the peripheral to the correct mode.
2) Giving the peripheral the chance to inform the host of any events that may require attention by the host.
3) Assuring the peripheral that the host is still functioning correctly and that it is OK to accept money from the customers.

4.2.1 Normal Mode Timing

Parameter	Lower Limit	Upper Limit	Description
Inter-character Timing	0ms	20ms	If the maximum inter-character time is exceeded, the receiver of the packet should discard the current packet and resume its search for an STX character.
Peripheral Response Time	0ms	35ms	If the peripheral device has not started to respond within this amount of time after a packet is sent to it, the host should assume that no reply is going to be sent by the peripheral.
Polling Interval	50ms	1s	This is the rate at which the host should communicate with the peripheral. The recommended polling interval is 200ms.
	50ms	250ms	BNF When a bunch note feeder is installed, a higher polling rate is required to maintain system performance.
Peripheral Disable Timeout	3s	-	If the peripheral does not receive a poll from the host within this time period, it will be disabled to avoid continuing to accept money on a system with a disabled host.

4.3 Special (Interrupt) Mode

Deprecated

In special mode, EBDS adds a facility for the peripheral device to request that the host perform a poll operation. The host is still responsible for running transactions to:

1) Sending any commands the host needs to set the peripheral to the correct mode.
2) Assuring the peripheral that the host is still functioning correctly and that it is OK to accept money from the customers.

When the Peripheral detects an event that requires host attention, it sends a single ENQ (0x05). The ENQ character will never be sent while a reply is being sent, but it can be sent while a host command is being sent. When the host receives the ENQ, it should poll the peripheral.

Note that while events usually follow a certain order, it is unwise to assume that certain events will always follow in a predictable manner. Thus each ENQ should be treated as if any or even no event could have been the trigger.

In addition, the host code must be written with the awareness that an ENQ can be sent at any time, without regard for other EBDS activity. For example, the device could send an ENQ in the middle of the host sending a command.

4.3.1 Special Mode Timing

Parameter	Lower Limit	Upper Limit	Description
Inter-character Timing	0ms	20ms	If the maximum inter-character time is exceeded, the receiver of the packet should discard the current packet and resume its search for an STX character.
Peripheral Response Time	0ms	35ms	If the peripheral device has not started to respond within this amount of time after a packet is sent to it, the host should assume that no reply is going to be sent by the peripheral.
Polling Interval	50ms	30s	This is the rate at which the host should communicate with the peripheral. The recommended polling interval is 1s.
Peripheral Disable Timeout	33s	-	If the peripheral does not receive a poll from the host within this time period, it will be disabled to avoid continuing to accept money on a system with a disabled host.
ENQ Response Time	0ms	100ms	If the host does not respond to the ENQ within the required time, the peripheral will retry sending the ENQ.
ENQ Response Timeout	3s	-	If an ENQ is sent and the host does not reply by this limit, the peripheral will be disabled to avoid continuing to accept money on a system with a disabled host.

The commands used to control the data link layer modes are covered in section 7.1.1

4.3.2 ABDS and Special (Interrupt) Mode

Incompatible

Special Mode may not be used in a system where ABDS (see section 5.2) is in use. This is due to the fact that multiple devices could all try to send an ENQ character at the same time and the result would be data corruption. The half-duplex, multi-drop character of the RS-485 physical layer of ABDS makes it incompatible with special (interrupt) mode.

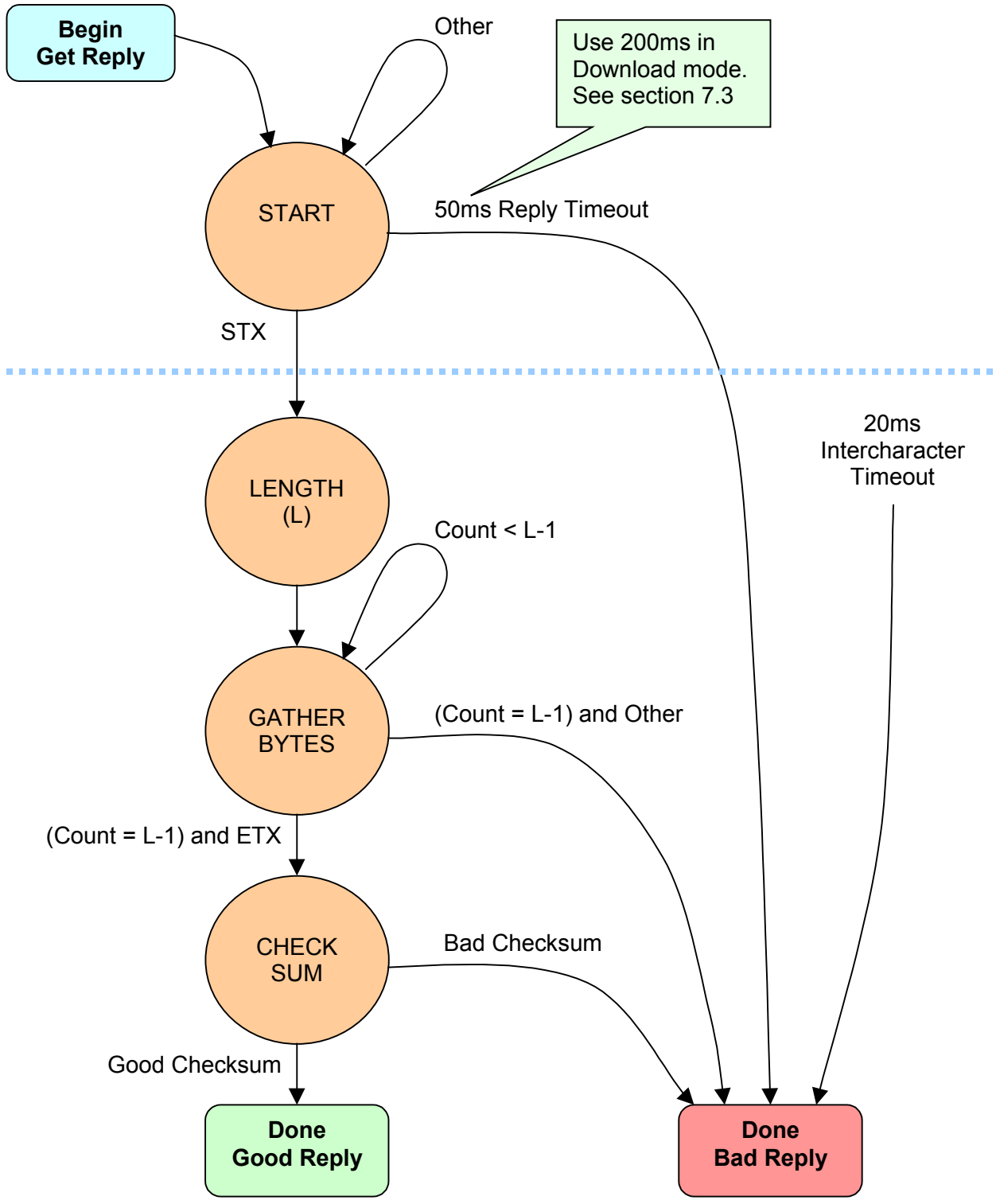
4.3.3 Flash Download and Special (Interrupt) Mode

Incompatible

Special Mode may not be used in conjunction with the Flash Download protocols (see section 7.3). If the ability to update device firmware via EBDS is desired, then Special Mode should not be employed.

4.4 Receiving a reply

The following is a state transition diagram for host code to receive a reply packet. This model embodies the best known practices for this task.



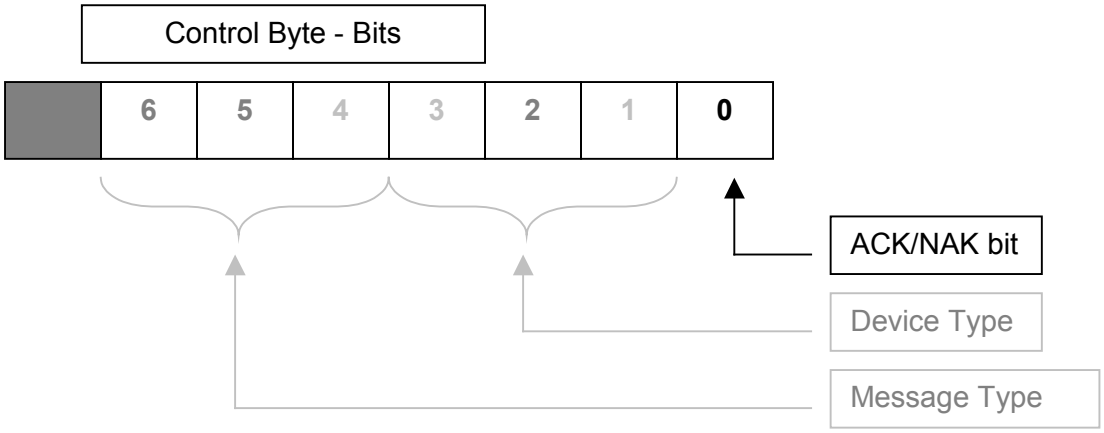
Reply Packet Receive State Diagram

4.4.1 Loopback Error Handling

Under some conditions, the host may encounter a fault in which data sent to the device is echoed or looped back to the host. To handle this, the host should mark as bad, any packet that exactly matches the packet that was sent by the host to the device. Reliably detecting this sort of error will reduce all sorts of issues in the field. The author speaks from personal experience in regards to this matter.

4.5 ACK/NAK Processing

In an EBDS transaction either side may decide to acknowledge the transfer by sending an ACK, or to deny acknowledgment by sending a NAK. This information is transmitted in the control byte. The control byte is laid out as follows:

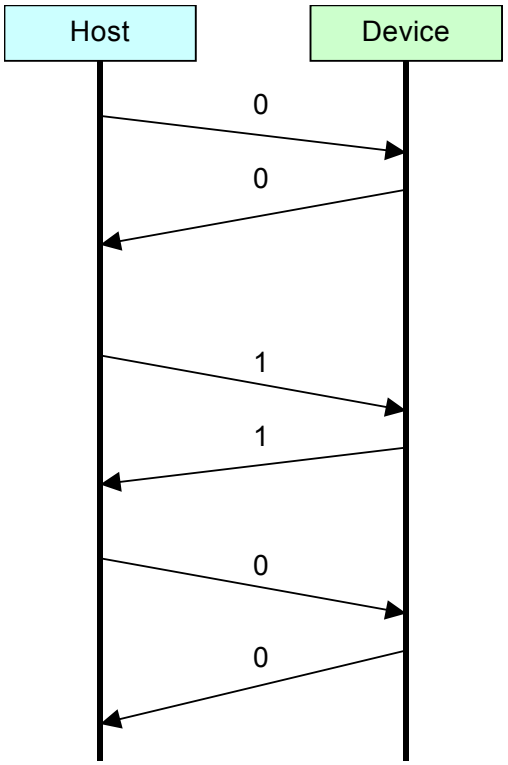


The ACK/NAK functionality is embedded in the matching of the bit sent by the host, compared with the bit sent by the device. In general, host systems indicate an ACK by toggling their ACK/NAK bit between transactions, and indicate a NAK by using the same value twice in a row. Devices indicate an ACK by matching the host's ACK/NAK bit and a NAK by returning a value different from the host's ACK/NAK bit.

This is illustrated in the next section.

4.5.1 ACK/NAK Examples:

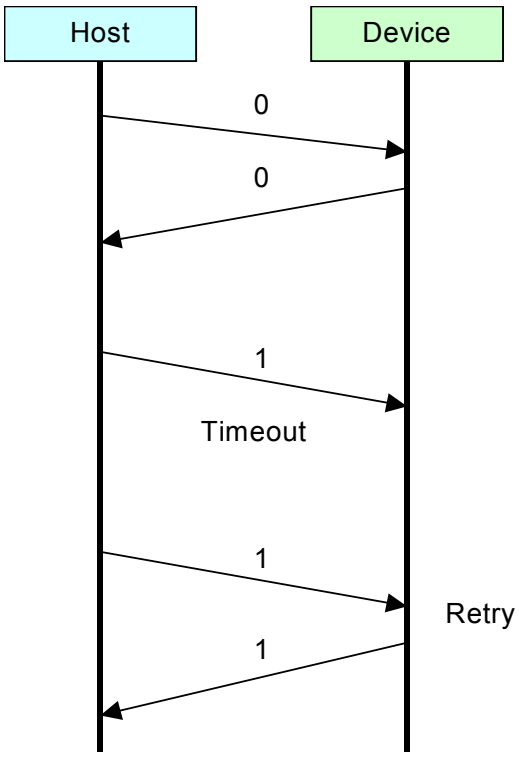
Under normal circumstances, the host alternates the value of the ACK bit in the control byte sent to the device and the device returns a matching value in its ACK bit in the reply. This is illustrated in this example:



Sometimes, the device is unable to respond to the host command. In these cases, the host resends the command with the same ACK value. This is illustrated in the first example.

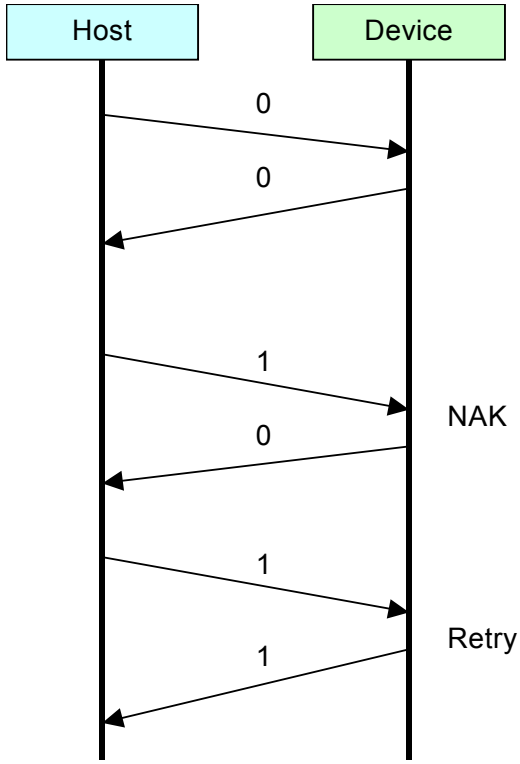
Older devices are more prone to this scenario as they have less processing "power", however all devices will from time to time be unable to respond to a host packet. The host system should retry the command. See the "Non-acknowledge Timeout" parameter in section 4.5.2 for details.

Note: This case is equivalent to the cases where either the host command or device reply is lost in transit.



In this example, the device is not ready to process the packet and explicitly sends a NAK to indicate this to the host. The host then retries and the command is accepted.

Note: This case also occurs during flash download where a simple retry is NOT sufficient. See section 7.3 for more details.



4.5.2 ACK/NAK Timing Requirements

In general, an ACK to a response can wait until the next scheduled poll and no special timing is required. A NAK however is subject to a special limit. Under some conditions, the host must NAK the device in fewer than 100ms to ensure the device does not proceed with money processing. If more than 100ms elapse, the device will proceed on the assumption that the host ACK is on its way. Since there is not way to be certain when these conditions exist the following spec applies to all NAK replies:

Parameter	Lower Limit	Upper Limit	Description
Initial NAK Response Time	50ms	100ms	When a NAK condition is detected, the host should communicate with the peripheral in less than 100ms.
Following NAK Response Time	50ms	3s	If subsequent NAKs are required, this more relaxed timing may be used.
Non-acknowledge Timeout	3s		If a device does not eventually respond, the host should consider the device to be out of service.

5. The Network Layer

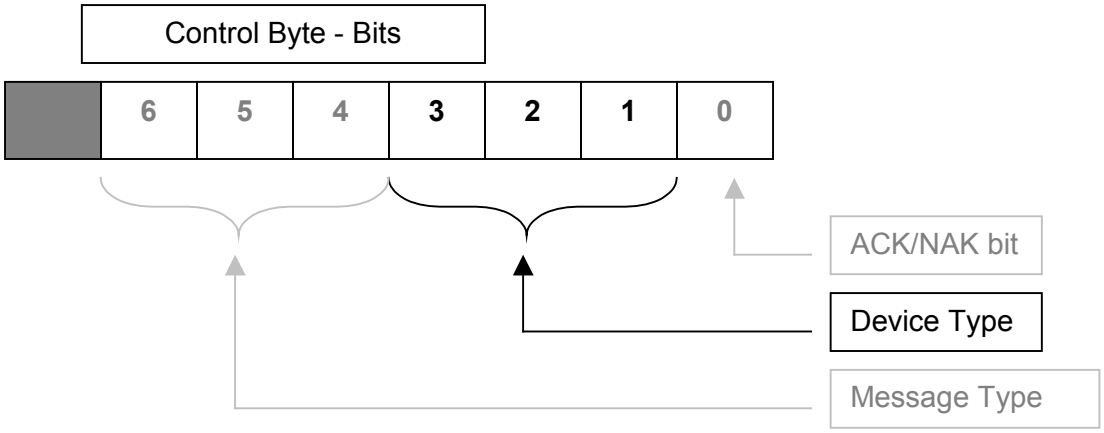
This section discusses the parts of the EBDS protocol that deal with the routing of packets to the correct device. There are two protocols for doing this: Device Type Routing a (proposed) protocol for routing based on the type of device and ABDS, a extension to EBDS to support multi-drop connection of multiple bill acceptors to a single host port.

5.1 Device Type Message Routing

From its inception, EBDS has been a protocol used to control bill acceptors. Over the years, the need has existed to support different types of transaction hardware (such as coin acceptors, smart card readers, and bill recyclers) with EBDS. This proposed protocol extension would allow for new device types to be added with harm to any existing host code.

Device type message routing is a method of sending commands to different sorts of devices in a single system. Packets sent by the host are sent to specific types of devices. Devices ignore messages not addressed to them. In reply packets, devices transmit their device type back to the host for confirmation.

The device routing of EBDS is encoded by the control byte. The device type field in this byte is used to identify the type of device intended to receive a command and to identify the type of device generating the reply.



The Device Type field of the control byte is decoded as follows:

Bit 3	Bit 2	Bit 1	Value	Description
0	0	0	0	Bill Acceptor with single escrow.
0	0	1	1	Reserved for Future Usage.
0	1	0	2	Reserved for Future Usage.
0	1	1	3	Reserved for Future Usage.
1	0	0	4	Reserved for Future Usage.
1	0	1	5	Reserved for Future Usage.
1	1	0	6	Reserved for Future Usage.
1	1	1	7	Reserved for Future Usage.

Device type message routing does not imply that the devices are connected in a multi-drop bus as in ABDS. In general, the host will have to determine the type of device connected to each port. This is done by attempting communications with the different device types starting with 0 and proceeding upward until a device responds. Once a device responds, the host should validate the device type in the response field to confirm the type of device attached.

5.2 ABDS Message Routing

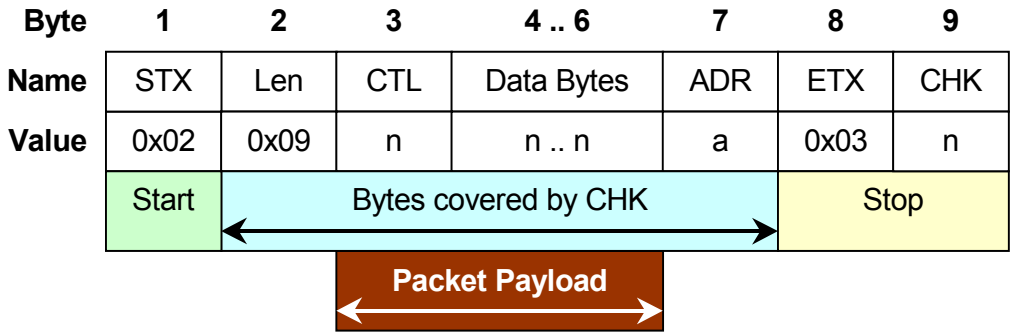
CFSC **S3K**

The ABDS message routing protocol is designed to handle multiple bill acceptors on a single, multi-drop, RS-485 data line. In theory, up to 31 devices can be attached to a single line, however, in practice far fewer (typically 2 through 5) are feasible. The RS-485 physical layer is required because it is the only interface that supports the required multi-drop capability. While still a current interface, ABDS routing is considered an older approach. In more modern systems, multiple bill acceptors are best accommodated with USB connections.

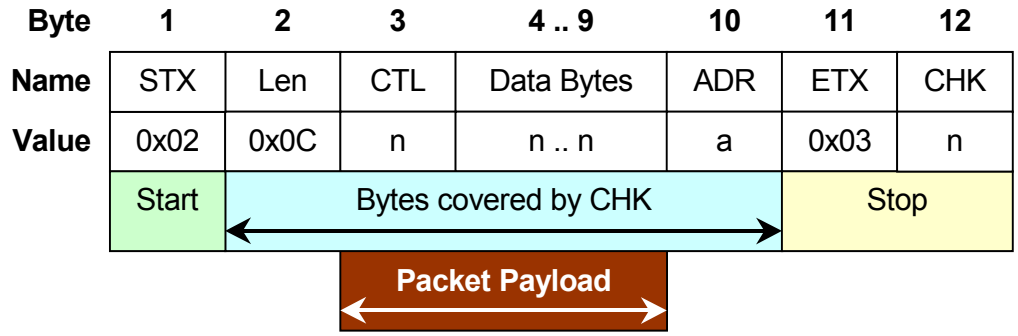
The number of allowed devices is limited by the need to poll all of the units in the chain in a timely manner. For example, if a polling rate of 200ms is desired then the maximum number of devices that can be attached is 200ms/50ms or about 4.

ABDS is an extension of EBDS that is identified by the use of specially laid out packets. The design of these packets for the host and the device are illustrated below.

The format of an ABDS command from the host is:



The format of an ABDS response from the device is:



Where:

ADR	The address of the device. (1..31, 0x01..0x1F). This corresponds to the binary address value set in the device's DIP switches.

5.2.1 ABDS Timing Requirements

Since ABDS is implemented on a shared RS-485 bus, it is necessary to control which party has its bus "drivers" turned on. To avoid bus conflicts, the bus driver must be turned on no more than 1ms before transmission begins and must be turned off no later than 1ms after the transmission is completed.

Parameter	Lower Limit	Upper Limit	Description
Bus Driver Turn-On Lead Time	0	1ms	The RS-485 Drivers must be turned on before data can be sent, but must not be turned on more than 1 ms before the start of data transmission.
Bus Driver Turn Off Hold Time	0	1ms	The RS-485 Drivers must not be turned off before the end of data transmission, but they must be off by no more than 1ms later.

5.2.2 ABDS and Special (Interrupt) Mode

Incompatible

Special Mode may not be used in a system where ABDS is in use. This is due to the fact that multiple devices could all try to send an ENQ character at the same time and the result would be data corruption. The half-duplex, multi-drop character of the RS-485 physical layer of ABDS makes it incompatible with special (interrupt) mode.

6. The Session Layer

In EBDS there is no concept of a login or logout or other such concepts normally associated with a session layer. None the less, session management is crucial to proper handling of the customer's money. In general, a session begins when the host program initiates communications with the device and ends when that communications is terminated. This is complicated by the fact that the host and the device may not be in synch. One can be shut down without the other being affected:

The application is responsible for handling its own start-up and shutdown issues. The device informs the application of its session status through the power-up event. When the host detects this event, it should handle session start issues with the bill acceptor. *So far, all attempts to have the device send a power-down event after power-loss have been without success.*

The control of the power up of the bill acceptor is detailed in section 8.1.

6.1 Variations in Power Up reporting

There exist in the field, two separate methods for a bill acceptor device to report that a start-up event has been detected:

S1K **CFMC** **CFSC**

These devices continue to report the power-up status until the host has completed dealing with any banknotes that may have been in the unit at the time.

S2K **S3K**

These devices report the power-up event until they are ready to process commands from the host.

It will be noted that these two approaches are incompatible and that the host code can either:

- Code for the power up response of the devices attached to the host.
- Based on the device connected apply the appropriate algorithm.
- Wait up to three seconds for the power-up status to clear before proceeding to issue commands to the bill acceptor.

6.2 Starting in Download mode

It is possible that the device will be in flash download mode on startup. There are a number of ways that this may transpire:

- The device may be a blank unit, simply lacking in application code.
- The host may have been downloading code to the unit, and while doing so, the host was reset, rebooted or restarted. The device is still in download mode.

- The host may have been downloading code to the unit, and while doing so, the entire system was powered down. The device is still in download mode.

When this occurs, the host has two distinct courses of action. It can:

1. Go out of service. This is the course of action expected from hosts that do not support flash download of the device.
2. Resume download of the device firmware. To facilitate this, the host should store the path and filename of the device firmware file in non-volatile storage so that it may be accessed if the session opens in download mode.

For more information on download mode and detecting download mode please see sections 7.1.2 and 7.3.

6.2 Ending the Session

When the host system prepares to terminate operations, it should ensure that the device is placed into a safe mode so that does not continue to operate without an application or retain currency after the application closes. To this end, the application should ensure that:

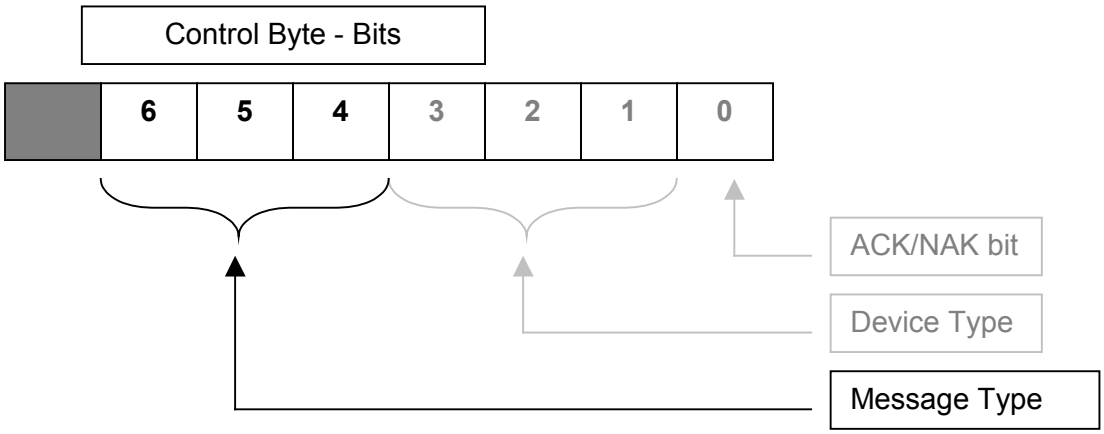
- Any currency currently in process (escrow) is returned to the consumer.
- The device is fully disabled so that no further currency will be accepted.

See sections 7, 8, and 9 for further details on the commands used to accomplish this.

7. The Presentation Layer (for Bill Acceptor)

The presentation layer of EBDS is specific to each type of device being supported. At present, only a bill acceptor device with a single note escrow is defined in EBDS.

The presentation layer of the bill acceptor is driven from the message type field in the control byte. This three-bit field determines how the packet is processed.



Bit 6	Bit 5	Bit 4	Value	When sent by the host	When sent by the device
0	0	0	0	*Reserved*	*Reserved*
0	x	1	1/3	Omnibus Command	*Not Used*
0	1	0	2	*Not Used*	Omnibus Reply
1	0	0	4	Calibrate Request	Calibrate Reply
1	0	1	5	Firmware Download Request	Firmware Download Reply
1	1	0	6	Auxiliary Command Request	Auxiliary Command Reply
1	1	1	7	Extended Commands	Extended Command Reply or Extended Omnibus Reply

7.1 The Omnibus Command

Omnibus, adj.:
Including or covering many things or classes: *an omnibus trade bill.*

The concept behind the omnibus command is simple. The host sends a packet with virtually everything needed to control a bill acceptor to the device, and the device responds with a packet with virtually everything needed by the host. Thus in theory, only one command is needed. In practice, the sophistication of the command set long ago reached the point where it was not feasible to fit in all the data all the time. Thus the auxiliary and extended commands were created. Despite this, the omnibus command remains the very core of EBDS and the most frequently used command. When one speaks of “polling” the bill acceptor, it is this command that is being referred too.

7.1.1 Omnibus Command

The layout of an omnibus command from the host is (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data 0	Data 1	Data 2
Value	1x/3x	nn	nn	nn

Omnibus Command – CTL Byte			
Bit #	**Name**	**Value**	**Description**
0	ACK	X	ACK/NAK value. See section 4.5 for details.
1	-	0	Must be 0
2	-	0	Must be 0
3	-	0	Must be 0
4	-	1	Must be 1
5	Bookmark		**Non-Retail**. A bookmark is a piece of paper stacked by the acceptor as an event marker in the cash box. For example a dispute with the customer. In some applications, especially those with lockable cassettes it may not be feasible to examine the cash box immediately. A bookmark may be used to mark the spot of the dispute for later examination. ➢ **CFSC**: The dimensions of the bookmark are 2.6" by 4" ➢ **S1K**, **S2K**, **S3K**, **CFMC**: The dimensions of the bookmark are 2.6" by 5"
		0	Disabled. ***Recommended.***
		1	Enabled. ***Warning***: Leaving bookmark enabled may cause folded bank notes to be treated as bookmarks resulting in lost credit to the customer. In addition, some currencies have valid bank notes that are very close in dimension to a bookmark. This raises the risk of a valid note being "stolen".
6	-	0	Must be 0

Omnibus Command – Data Byte 0 (Terse Note Reporting)			
Bit #	**Name**	**Value**	**Description**
0 **1** **2** **3** **4** **5** **6**	Denom1 Denom2 Denom3 Denom4 Denom5 Denom6 Denom7	0	Disable Denomination n
		1	Enable Denomination n

Omnibus Command – Data Byte 0 (Expanded Note Reporting)			
Bit #	**Name**	**Value**	**Description**
0..6	Bills	0x00	All bills are disabled.
		Other	Bills are enabled based on the extended note enable settings.

Note that denominations may also be disabled through the configuration of the bill acceptor (by either configuration coupon or on some models, "DIP" switches). In that case, the bill will remain disabled, even if the EBDS command to enable is sent.

Omnibus Command – Data Byte 1			
Bit #	**Name**	**Value**	**Description**
0	Special Mode	Data Link Layer mode control. See sections 4.2 and 4.3 for details on Normal (Polled) mode and Special (Interrupt) modes.	
		0	Normal (Polled) Mode. ***Recommended.***
		1	Special (Interrupt) Mode. **Deprecated**
1	High Security	This field controls the validation criteria are applied to bank notes.	
		0	Accept bills in high acceptance mode. ***Recommended.***
		1	Accept bills in high security mode. Note that acceptance of valid notes may suffer.
2..3	Orientation Control	This field controls the acceptance of bank notes based on the orientation of those notes as they enter the bill acceptor. Note that bill orientations can also be controlled by a configuration coupon or on some models, "DIP" switches. In all cases, the most accommodating of the settings is used. See sections 8.2.4 and 8.2.5 for more details on controlling the orientation of bill acceptance.	
		00	1-way: Accept bills fed right edge first, face up only.
		01	2-way: Accept bills fed face up only.
		1x	4-way: Accept bills fed any way.
4	Escrow Mode	This mode determines how bills are handled after the bills have been validated. Note that bills that are unable to be validated are always rejected.	
		0	Escrow Mode is disabled. (Non-escrow mode) **Deprecated** When the bill acceptor validates a bill, it immediately stacks the note in the cash box. The host only receives notification when the note is stacked.
		1	Escrow Mode is enabled. ***Recommended.*** When the bill acceptor validates a bill, it informs the host of the bill by sending an escrow event. The host then decides if the bill should be stacked or returned to the consumer. See section 8.2.3 for best practices in bill handling.
5	Bill Stack Command	0	No operation.
		1	If a bill is in escrow, stack it in the cash box. Note that this command is only valid if Escrow mode is enabled and a bill is in escrow. This command and the Bill Return command are mutually exclusive.

6	Bill Return Command	0	No operation.
		1	If a bill is in escrow, return it to the consumer. Note that this command is only valid if Escrow mode is enabled and a bill is in escrow. This command and the Bill Stack command are mutually exclusive.

Omnibus Command – Data Byte 2

Bit #	Name	Value	Description
0	NoPush Mode		S2K, S3K: There are times when the bill acceptor is unable to give credit for a note due to a problem in transporting the bill, and the bill cannot be returned to the customer. In these cases, this bit determines how such bills should be handled.
		0	Push non-credit notes into the stacker and continue operating. ***Recommended.***
		1	Do not push non-credit notes. Stall the bill acceptor with the note still in the bill path. A manager/technician level intervention is required. In most retail applications, taking the system out-of-service is not a desirable option.
1	Bar Code		Non-Retail, S1K, CFMC, CFSC. A bar-coded voucher is a document with a unique bar-coded identity number encoded into it. These identity numbers are referenced against an external database by the host to determine the validity and value of the voucher. Notes: Bar code vouchers must be inserted “face up”. Bar coded vouchers can only be processed if Escrow mode is enabled.
		0	Bar-coded vouchers are disabled.
		1	Bar-coded vouchers are enabled. Bar coded vouchers are reported to the host via the Expanded Omnibus Bar Code Reply packets. See section 7.1.3 for details.
2 **3**	PUP-B PUP-C		S1K, CHMC, CFSC. These bits are used to control the behavior of the bill acceptor when a bill is found in the bill path during power up. See section 6.1 for more information on power up issues.

	Bill Position	Pre-Escrow	Escrow	Post-Escrow
00	Power Up Policy – A:	Return bill	Wait for host with unknown value.	Stack with unknown value.
01	Power Up Policy – B:	Return bill	Go out of service	Stack with note actual value.
10	Power Up Policy – C:	Return bill	Go out of service	Stack with unknown value.
11	Reserved.	-	-	-

4	Expanded Note Reporting	There are two methods of reporting the value of banknotes validated/stacked by the bill acceptor. For more details on handling bill values see section 8.2.	
		0	Use terse note reporting. Notes are reported as the generic Denom1 though 7. ➢ Bills are enable/disabled via the Denom1 through Denom7 bits in Omnibus Command – Byte 0 ➢ Bills are reported to the host via the three bit Denomination field in Omnibus Reply – Byte 2.
		1	**CFSC** Use expanded note reporting. ➢ All bills may be disabled via the Bills field in Omnibus Command – Byte 0. ➢ Bills are enabled/disabled via the Set Note Inhibits command. See section 7.5.2 below for details. ➢ Bills are reported to the host via the Expanded Omnibus Bill Reply packets. See section 7.1.4 for details.
5	Expanded Coupon Reporting	0	No special handling of generic coupons. MEI™ Generic Coupons (if supported) are reported the same as a bank note of the same value. Free vend coupons are not supported.
		1	**S2K-US**, Enable detailed reporting of MEI™ Generic Coupons. The host receives details on the type and identification of generic coupons fed into the bill acceptor. See section 7.1.5 for details.
6	Reserved	0	Currently 0, RFU

7.1.2 Standard Omnibus Reply

The most common reply to an Omnibus Command is the standard reply. However, if either bar coded vouchers or extended note reporting are enabled, then other reply formats are possible. See sections 7.1.3 and 7.1.4 for details on these replies respectively.

The standard reply to an omnibus command take the following form (STX, Length=0x0B, ETX, and CHK omitted).

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	2n	nn	nn	nn	nn	nn	nn

The CTL byte conveys no data beyond identifying the type of reply and is not further examined.

Data byte 0 is used to describe the current state or activity of the bill acceptor. This is accomplished through a bit map of events, states and conditions, listed below:

Omnibus Reply – Data Byte 0			
Bit #	**Name**	**Value**	**Description**
0	*Idling*	1	The bill acceptor is idling between bill transactions.
1	*Accepting*	1	The bill acceptor is drawing in a bill.
2	Escrowed Event	1	There is a valid bill in escrow.
3	*Stacking*	1	The bill acceptor is stacking a bill.
4	Stacked Event	1	The bill acceptor has stacked a bill.
5	*Returning*	1	The bill acceptor is returning a bill to the customer.
6	Returned Event	1	The bill acceptor has returned a bill to the customer.

There are a few points that bear further examination.

- If all seven bits are 0, then the bill acceptor is out of service.
- All of the bits ending in "ing" are transient status conditions. That means that depending on polling rate, bill feed rate, or just plain luck, these states may or may not be "seen" by the host system. That is why the use of these bits to drive any host actions is **deprecated**.

Omnibus Reply – Data Byte 1			
Bit #	**Name**	**Value**	**Description**
0	Cheated	1	The unit has detected conditions consistent with an attempt to defraud the system. This may also occur when there is a problem transporting the bill to the cash box. MEI does not offer a method to set / test cheat events.
1	Rejected	1	The document presented to the bill acceptor could not be validated and was returned to the customer.
2	Jammed	0	The bill path is clear.
		1	The bill path is blocked and the bill acceptor has been unable to resolve the issue. Intervention is required.
3	Stacker Full	0	Normal operation
		1	The cash box is full of bank notes and no more may be accepted. Intervention is required.
4	Cassette Attached	0	The cash box has been removed. No bills may be accepted.
		1	The cash box is attached to the unit.
5	Paused	1	**S2K**, **S3K**. The customer is attempting to feed another note while the previous note is still being processed. The customer must remove the note to permit processing to continue.
6	Calibration in progress	It is possible to field calibrate bill acceptors. In general, due to advances in processes used in manufacturing and continuous self-calibration, this is not needed. Calibrating a unit with an incorrect document will greatly reduce performance. For more information on field calibration please refer to section 7.2	
		0	Normal operation
		1	The unit is in calibration mode. Intervention is required to feed a special calibration document into the bill acceptor.

Omnibus Reply – Data Byte 2			
Bit #	**Name**	**Value**	**Description**
0	Power Up	1	The bill acceptor has been powered up. The host must deal with any notes in escrow, and reinitialize the bill acceptor to the desired operating settings.
1	Invalid Command	1	The bill acceptor received an invalid command.
2	Failure	1	The bill acceptor has encountered a problem and is out of service. Intervention is required.
3..5	Bill Value	The terse bill value field. This field is valid when the bill acceptor is in terse mode and either the escrow or stacked bits are set. (See Omnibus Reply – Data Byte 0 for details of those events)	
		000	Unknown/No credit. This condition is returned for a wide variety of reasons: ➢ When a bookmark is either escrowed or stacked. ➢ When a bar coded voucher is escrowed or stacked. ➢ When a cheated or jammed document is stacked. ➢ When a jammed document is returned to the consumer. ➢ In extended note mode, a note is in escrow or stacked. ➢ On power up or cash box install when the unit performs a "run & stack" action.
		001	Denom1 – Typically $1
		010	Denom2 – Typically $2
		011	Denom3 – Typically $5
		100	Denom4 – Typically $10
		101	Denom5 – Typically $20
		110	Denom6 – Typically $50
		111	Denom7 – Typically $100
6	-	0	Currently 0, RFU

Omnibus Reply – Data Byte 3			
Bit #	**Name**	**Value**	**Description**
0	Stalled	1	The bill acceptor is stalled (in NoPush mode), with a bank note still in the bill path.
1	Flash Download	1	A flash download is ready to commence. The host may begin send download records. See section 7.3 and @ for details.
2	Pre-stack	1	**Non-Retail**, **Deprecated**. This bit indicates that the bill has reached a point in the stacking process where it can no longer be retrieved.
3	Raw Barcode	0	24 character barcodes will be converted to 18 character barcodes.
		1	24 character barcodes will not be converted to 18 character barcodes.
4	Device Caps	0	The Query Device Capabilities command is not supported. (*1)
		1	The Query Device Capabilities command is supported.
5	-	0	Currently 0, RFU
6	-	0	Currently 0, RFU

Note: On some implementations of EBDS, the Device Caps bit is suppressed to maintain compatibility with non-compliant host systems. See section 7.4.14 for more details.

Omnibus Reply – Data Byte 4			
Bit #	**Name**	**Value**	**Description**
0..6	Model Number	nn	A value that represent the model of the bill acceptor. This is interpreted in the tables below.

Series 1000	**Hex**	**Decimal**	**ASCII**	**Product**
	0x01	1		Discontinued, ZT1000, US
	0x0C	12		Discontinued, ZT1107, US
	0x0F	15		Discontinued, ZT1200, Australia
	0x14	20		Obsolete, ZT1200, US

Series 2000	**Hex**	**Decimal**	**ASCII**	**Product**
	0x41	65	**A**	AE2600 Gen2D, Australia
	0x42	66	**B**	AE2800 Gen2D, Russia
	0x43	67	**C**	AE2600 Gen2D, Canada
	0x44	68	**D**	AE2800 Gen2D, Euro
	0x45	69	**E**	Reserved (VN2300 US Economy)
	0x46	70	**F**	Reserved (VN2600 Gen2B & 2D, China)
	0x47	71	**G**	Reserved (AE2800 Gen2D, Argentina)
	0x4D	77	**M**	AE2800 Gen2D, Mexico
	0x50	80	**P**	AE2600 Gen2B, C and D, US Premium
	0x51	81	**Q**	Discontinued, Philippines
	0x57	87	**W**	AE2800 Gen2D, Brazil
	0x58	88	**X**	AE2800 Gen2D, US Expanded

Series 3000	**Hex**	**Decimal**	**ASCII**	**Product**
	0x1E	30		Discontinued, Series 3000 VFX (BDS)
	0x1F	31		Obsolete, Series 3000 EBDS

Casflow SC	**Hex**	**Decimal**	**ASCII**	**Product**
	0x4A	74	**J**	Discontinued, Cashflow SC 66, Monolithic Code
	0x54	84	**T**	Cashflow SC 83, Split Component/Extended Notes
	0x55	85	**U**	Cashflow SC 66, Split Component/Extended Notes

Omnibus Reply – Data Byte 5			
Bit #	**Name**	**Value**	**Description**
0..6	Code Revision	nn	The version number of the firmware code in the bill acceptor. This may be coded as: ➢ **S1K**, **S3K**, **CFMC**, **CFSC**: A seven bit binary value with an implied divide by 10. (versions 0.0 through 12.7) ➢ **S2K**: A 1 ¾ digit BCD value with an implied divide by 10. (versions 0.0 through 7.9).

Note that in general, the version number of the code is not sufficient to identify that software. This because different software parts use independent version numbers. Version numbers are

only useful for comparing firmware from the same software part. See section 8.3 for more details on determining the software in a unit.

7.1.3 Extended Omnibus Bar Code Reply

S1K **CFMC** **CFSC** **Non-Retail**

If bar coded vouchers are enabled, then the bill acceptor will send this reply when a bar-coded voucher is in escrow. This reply contains an additional 28 bytes of ASCII encoded decimal bar code data. Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. This data ends when the first 0x28, ASCII "(" character occurs. The following is the layout of this packet (STX, Length=0x28, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4
Value	0x7n	0x01	nn	nn	nn	nn	nn

10	**11**	**12**		**38**
Data 5	Ext Data 0	Ext Data 1	**o o o**	Ext Data 27
nn	nn	nn		nn

A typical data payload for an 18 digit bar-coded voucher might be encoded as:

```
012345678901234567((((((((((
```

It is then the responsibility of the host system to determine what (if any) value to assign to this voucher. The host must command the bill acceptor to either stack or return the voucher. It is important to note that when the voucher is stacked, a standard omnibus reply with a terse bill value of unknown is sent by the bill acceptor. Thus, the only chance the host system has to capture the bar coded data is when the voucher is at escrow.

Note: On some retail versions of the Cashflow-SC product line, support for Bar Coded documents has been removed.

7.1.4 Extended Omnibus Expanded Note Reply

CFSC

If expanded note reporting is enabled then the bill acceptor will send this reply when a bank note is in escrow or is stacked. The reply contains 18 additional bytes of data that describe the bank note in great detail. Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. The following is the layout of this packet (STX, Length=0x1E, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4
Value	0x7n	0x02	nn	nn	nn	nn	nn

10	11	12		28
Data 5	Ext Data 0	Ext Data 1	o o o	Ext Data 17
nn	nn	nn		nn

The extended bill data is described below:

Field	Byte Offset	Field Description	Sample Value (2000 Yen Note)
Index	0	Not used for escrow or stacked notes	0x00
ISO Code	1..3	A three character ASCII currency code. See ISO 4217 for details	"JPY"
Base Value	4..6	A three character ASCII coded decimal value	"002"
Sign	7	An ASCII coded sign value for the Exponent. This field is either a "+" or a "-"	"+"
Exponent	8..9	ASCII coded decimal value for the power of ten that the base is to either be multiplied by (if Sign is "+") or divided by (if Sign is "-")	"03"
Orientation	10	A single character binary field that encodes the orientation of the bill. 0x00 = Right Edge, Face Up 0x01 = Right Edge, Face Down 0x02 = Left Edge, Face Up 0x03 = Left Edge, Face Down Note: In general, this field is only correct if the Extended orientation bit is set in device capabilities map. See section 7.4.14.	0x00
Type	11	An ASCII letter that documents the note type. This corresponds to the data in the variant identity card.	"A"
Series	12	An ASCII letter that documents the note series. This corresponds to the data in the variant identity card.	"A"
Compatibility	13	An ASCII letter that documents the revision of the recognition core used. This corresponds to the data in the variant identity card.	"B"
Version	14	An ASCII letter that documents the version of the note's recognition criteria. This corresponds to the data in the variant identity card.	"A"
Reserved	15..17	3 bytes reserved for future use.	N/A

In this example: Bill Value = 002 x 10^{+03} = 2 x 1000 = ¥2000 fed right edge first, face up.

A typical trace of the activity involved in processing a bill is illustrated below: (HOST / *DEVICE*)

A bill arrives at escrow (can you tell what type of bill? See the answer below)

```
02 08 10 7F  1C 12 03 69
02 1E 70 02  04 10 00 00  55 12 00 55  53 44 30 30
31 2B 30 30  00 43 41 42  42 00 00 00  03 65
```

The host commands the bill be stacked.

```
02 08 11 7F  3C 12 03 48
02 1E 71 02  04 10 00 00  55 12 00 55  53 44 30 30
31 2B 30 30  00 43 41 42  42 00 00 00  03 64
```

Stacking

```
02 08 10 7F  1C 12 03 69
02 0B 20 08  10 00 00 55  12 03 74
```

Stacking

```
02 08 11 7F  1C 12 03 68
02 0B 21 08  10 00 00 55  12 03 75
```

Stacking

```
02 08 10 7F  1C 12 03 69
02 0B 20 08  10 00 00 55  12 03 74
```

Stacking

```
02 08 11 7F  1C 12 03 68
02 0B 21 08  10 00 00 55  12 03 75
```

Stacking

```
02 08 10 7F  1C 12 03 69
02 0B 20 08  10 00 00 55  12 03 74
```

Stacked

```
02 08 11 7F  1C 12 03 68
02 1E 71 02  11 10 00 00  55 12 00 55  53 44 30 30
31 2B 30 30  00 43 41 42  42 00 00 00  03 71
```

Answer: USD $1, Right Edge, Face Up, C, A, B, B.

A special case is that of an unknown item stack in extended note mode. In this case the entire eighteen bytes of extended note data are null or zero bytes. An example of such a packet is shown below:

Host – Command:

```
02 08 10 7F  1C 12 03 69
```

Device – Reply:

```
02 1E 70 02  11 10 00 00  55 12 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  03 2A
```

7.1.5 Extended Omnibus Expanded Coupon Reply

S2K-US

If expended note reporting is enabled then the bill acceptor will send this reply when a bank note is in escrow or is stacked. The reply contains 6 additional bytes of data that describe the coupon in detail. Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. The following is the layout of this packet (STX, Length=0x12, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x7n	0x04	nn	nn	nn	nn	nn	nn

11	**12**	**13**	**14**	**15**	**16**
Coupon 1	Coupon 2	Coupon 3	Coupon 4	Fill	Fill
0x0n	0x0n	0x0n	0x0n	00	00

The coupon 1 through 4 bytes represent a 16 bit value that may extracted as follows (using “C” style array indexing starting at 0 rather than 1).

```
CouponData = ((Reply[11] & 0x0F) << 12) +
             ((Reply[12] & 0x0F) <<  8) +
             ((Reply[13] & 0x0F) <<  4) +
             ((Reply[14] & 0x0F)   + 1);
```

This sixteen bit value may then be further broken down as:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	**Description**
													v	**v**	**v**	3 bit coupon value
													0	0	0	Free Vend
													0	0	1	$1 Coupon
													0	1	0	$2 Coupon
													0	1	1	$5 Coupon
													1	x	x	Reserved
n	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	**n**	13 bit vendor ID			

7.2 The Calibrate Command

Deprecated

All of the bill acceptors discussed in this document have the ability to perform a field calibration procedure. In this procedure the host issues a calibration command and a calibration document is fed into the bill acceptor. The calibration documents are specifically designed for this purpose and must be kept clean and unwrinkled. In particular the following must **NOT** be used as a calibration document:

- Bank note currency of any sort.
- Bits of blank paper or magazine articles cut to the shape and size of a bill.
- The calibration document of a different model bill acceptor.

Modern bill acceptors contain self-calibration routines that continually adjust and tune the recognition sub-system. Thus field calibration is seldom necessary. However, if a calibration document is required, it should be obtained from an authorized service center.

The layout of a calibrate command from the host is (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data 0	Data 1	Data 2
Value	0x4n	nn	nn	nn

The acceptor responds with a standard omnibus reply with a CTL value of 4x.

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x4n	nn	nn	nn	nn	nn	nn

When the acceptor is ready to begin the calibration process, it will set the Calibration bit in Data1, Bit 6 (see section 7.1.2 Omnibus Reply – Data Byte 1). After this bit is set the host should return to polling via the standard omnibus command (see section 7.1.1)

At this point the calibration document can be inserted into the unit. The document will be drawn in and returned. When removed from the unit, the calibration procedure will be completed. The acceptor will indicate that the calibration was successful by resetting itself and reporting a power-up event (see section 7.1.2 Omnibus Reply – Data Byte 2). If the calibration fails, the unit remains in calibration mode until manually reset.

Note: A calibrate command should not be attempted if the bill acceptor indicates that its status is anything other than “Idling” (see section 7.1.2 Omnibus Reply – Data Byte 0).

7.3 The Download Firmware Command

S1K **Gen2D** **S3K** **CFMC** **CFSC**

Most MEI Bill Acceptors can be upgraded with new software via the EBDS interface. Host support of the download process allows the bill acceptor to be updated to the current code level, without the necessity of manual intervention. The host can be remotely commanded to upgrade the units, saving both time and money.

7.3.1 The pace of communications during download.

Download mode is somewhat different in approach than other modes. Normally, transactions are sent at a rate that allows events from the bill acceptor to be processed without using up a lot of bandwidth. This is not the case in download mode. In download mode, a great deal of data needs to be sent to the bill acceptor. Thus it is expected that the host will send data as rapidly as the protocol allows.

Another consideration however is the fact that many devices are unable to process communications traffic while they are in the midst of programming their flash memory. To allow for this, the response timeout for device replies should be increased from 50ms to 200ms. This change to the receive reply algorithm is called out in section 4.4.

7.3.2 Overview of the Flash Download Process

The download process has three distinct phases: Starting, Downloading and Finishing.

Starting: The purpose of the starting phase is to get the device out of normal operation mode and into downloading mode. This is done by first polling the unit, with a command as shown below (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data 0	Data 1	Data 2
Value	0x1n	00	00	00

This is a standard omnibus command (as specified in section 7.1.1) with all acceptance and options off. There are two possible responses that may occur. If the unit is not currently in download mode, a standard omnibus reply will be sent. This is detailed in section 7.1.2. If the unit is already in download mode, the following response will be sent (STX, Length=0x09, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**
Name	CTL	Pkt #	Pkt #	Pkt #	Pkt #
Value	0x5n	0n	0n	0n	0n

The 16 bit packet number is encoded, four bits at a time, in bytes 4 through 5 (high nibble first). If the unit is already in downloading mode then the starting phase is completed. Otherwise it will be necessary to place the device into downloading mode. This is accomplished with the Start

Download command. This command is shown below (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data 0	Data 1	Data 2
Value	0x5n	00	00	00/10

The response to this command is (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x5n	XX	XX	XX	00/02	XX	XX

Where "XX" are ignored values and Data 3 contains the status of the unit. While the Flash Download bit (see section 7.1.2) is not set, the device is not yet in download mode and the Start Download command must be resent. When that bit is set, the unit is expecting the host to enter the downloading phase of the download process.

Downloading: In this phase, the new code is sent to the unit and programmed into flash memory. Starting at the start of the file, the host sends 32 byte blocks of data to the device (Note: the file is required to by a multiple of 32 bytes longs). This is done through the Download Data command show below (STX, Length=0x49, ETX, and CHK omitted):

Byte	3	4	5	6	7
Name	CTL	Pkt #	Pkt #	Pkt #	Pkt #
Value	0x5n	0n	0n	0n	0n

8	9		70	71
Data 1 Hi	Data 1 Lo	o o o	Data 32 Hi	Data 32 Lo
0n	0n		0n	0n

The reply to this command is the download in progress reply (already seen above) shown here (STX, Length=0x09, ETX, and CHK omitted):

Byte	3	4	5	6	7
Name	CTL	Pkt #	Pkt #	Pkt #	Pkt #
Value	0x5n	0n	0n	0n	0n

Crucial to the process is responding correctly to the devices ACK/NAK of the data.

- If the device ACKs the packet, the host should step to the next packet.
- If the device NAKs the packet, then the host needs to "resync" with the device. This is accomplished by changing the block number to the value contained in the reply plus one. This is shown below (using "C" style array indexing starting at 0 rather than 1).

```
ReplyBlockNum = (((Reply[3] & 0x0F) << 12) +
                 ((Reply[4] & 0x0F) <<  8) +
                 ((Reply[5] & 0x0F) <<  4) +
                 ((Reply[6] & 0x0F)   + 1)) & 0xFFFF;
```

If the device sends more than 10 consecutive NAKs, unexpected packets, checksum or other errors, the host should abort the download process. At this point, the device is likely out of service and intervention is required to restore normal operation.

Finishing: Once the last block of data has been sent to the device, the host must wait. There are two phases to this wait.

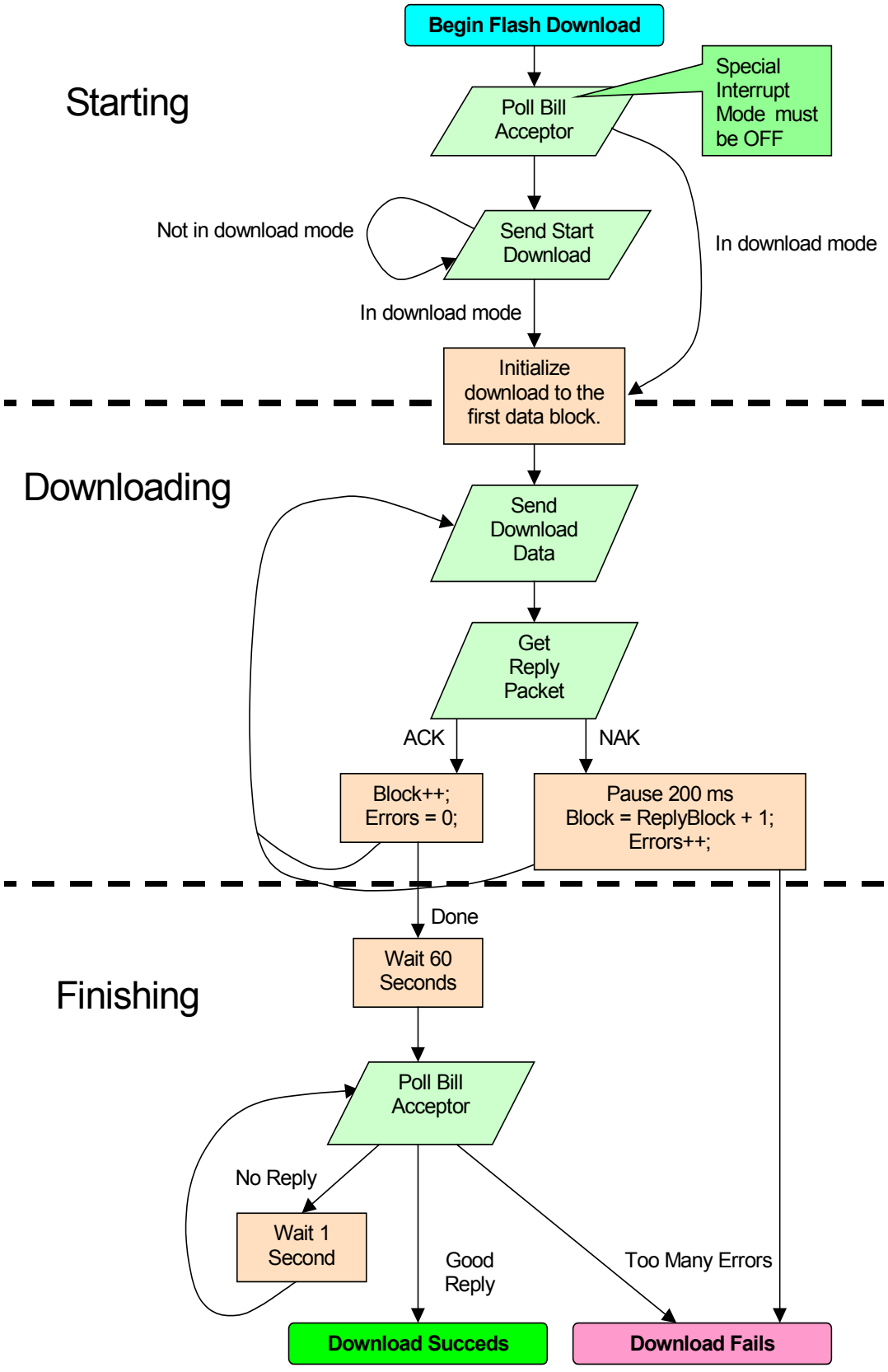
In phase one (passive), the host does nothing. It simply sits idle (at least with respect to the device being programmed) and waits. This lasts for at least fifteen seconds. After this time, the device will have also ended the download phase and the host can begin waiting for it to reboot and restart communications.

In phase two (active), the host slowly polls (about once a second) the device, waiting for it to reboot. When the device responds, the download process is complete. If a normal response was given by the device, then programming is complete and the device is ready to go back into service. If the unit is still in download mode at this point, it means that one of two scenarios exists:

1) The download failed for some reason. (Invalid file, Wrong kind of file, File/Device are incompatible, Errors in communication etc)
2) In devices with multi-part flash programs, the host needs to download the next file component of the device application. Currently on the Casflow-SC66, SC83 and SC85 products support multi-part flash (specifically two parts, Application and Variant).

7.3.3 Download Flow Diagram

The following is a flowchart summary of the download process:



7.4 The Auxiliary Commands

The Auxiliary Commands are used to provide functionality outside the scope of the Omnibus command(s) in the previous sections. These commands are somewhat code-base specific and no code base implements all of the commands, so be sure to check the compatibility icons before each section.

All auxiliary commands take the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	X	Y	Z

Where the Cmd field is the command value for the operation and Data A and Data B are arguments to that command. The supported commands are listed below:

Data A	Data B	Cmd	Description
0x00	0x00	0x00	Query Software CRC
0x00	0x00	0x01	Query Cash Box Total
0x00	0x00	0x02	Query Device Resets
0x00	0x00	0x03	Clear Cash Box Total
0x00	0x00	0x04	Query Acceptor Type
0x00	0x00	0x05	Query Acceptor Serial Number
0x00	0x00	0x06	Query Acceptor Boot Part Number
0x00	0x00	0x07	Query Acceptor Application Part Number
0x00	0x00	0x08	Query Acceptor Variant Name
0x00	0x00	0x09	Query Acceptor Variant Part Number
0x00	0x00	0x0A	Query Acceptor Audit Life Time Totals
0x00	0x00	0x0B	Query Acceptor Audit QP Measures
0x00	0x00	0x0C	Query Acceptor Audit Performance Measures
0x00	0x00	0x0D	Query Device Capabilities
0x00	0x00	0x0E	Query Acceptor Application ID
0x00	0x00	0x0F	Query Acceptor Variant ID
0x00	0x00	0x10	Query BNF Status
Bezel	0x00	0x11	Set Bezel
0x7F	0x7F	0x7F	Acceptor Soft Reset

All other values are reserved for future use.

7.4.1 Query Software CRC

S1K Gen2D S3K CFMC CFSC

This command is used to query the device for the 16 bit CRC of the flash contents. The query CRC command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	00

The reply from the device takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	CRC	CRC	CRC	CRC	XX	XX

Where the 16 bit CRC data is sent in Data 0 through Data 3, four bits at a time. This may be extracted as shown below (using “C” style array indexing starting at 0 rather than 1).

```
CRC_Value =  ((Reply[3] & 0x0F) << 12) +
             ((Reply[4] & 0x0F) <<  8) +
             ((Reply[5] & 0x0F) <<  4) +
             ((Reply[6] & 0x0F)      );
```

7.4.2 Query Cash Box Total

Gen2D

This command is used to query the amount of currency that has been counted going into the cash box. This count, stored in non-volatile storage, is represented as a 24 bit integer, though most hosts will store it as a 32 value. The query cash box total command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x01

The reply from the device takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	Data	Data	Data	Data	Data	Data

Where the Total Value is sent in Data 0 through Data 5. This may be extracted as shown below (using “C” style array indexing starting at 0 rather than 1).

```
Total_Value = ((Reply[3] & 0x0F) << 20) +
              ((Reply[4] & 0x0F) << 16) +
              ((Reply[5] & 0x0F) << 12) +
              ((Reply[6] & 0x0F) <<  8) +
              ((Reply[7] & 0x0F) <<  4) +
              ((Reply[8] & 0x0F)      );
```

NOTE: When the cash box is removed (see section 7.1.2, Cassette Attached) it is assumed that it is being emptied. Thus when this command is issued to the Bill Acceptor after the cash box is restored, the bill tally shall be returned to the host and then the tally shall be set to zero to begin counting bills in the now empty cash box.

If the host "knows" that the cash box was possibly removed while the system was off, it can use the Clear Cash Box Total command (section 7.4.4) to clear the tally.

7.4.3 Query Device Resets

Gen2D **CFSC**

This command is used to query the number of times the device has been reset. This is represented as a 24 bit integers, though most hosts will store it as a 32 value. The query device resets command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x02

The reply from the device takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	Data	Data	Data	Data	Data	Data

Where the Reset Count is sent in Data 0 through Data 5. This may be extracted as shown below (using "C" style array indexing starting at 0 rather than 1).

```
Reset_Count = ((Reply[3] & 0x0F) << 20) +
              ((Reply[4] & 0x0F) << 16) +
              ((Reply[5] & 0x0F) << 12) +
              ((Reply[6] & 0x0F) <<  8) +
              ((Reply[7] & 0x0F) <<  4) +
              ((Reply[8] & 0x0F)      );
```

7.4.4 Clear Cash Box Total

Gen2D

This command is used to reset the count of bills entering the cash box. The clear cash box total command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x03

No data is returned to the host in the reply, shown below form (STX, Length=0x19, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	00	00	00	00	00	00

7.4.5 Query Acceptor Type

CFSC

This command is used to determine the type of bill acceptor installed. The query acceptor type command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x04

The data returned by the device takes the form of a ASCII string that is either 20 bytes long or is terminated by a non-printable character. The reply packet is shown below (STX, Length=0x19, ETX, and CHK omitted):

Byte	**3**	**4**	**5**		**22**	**23**
Name	CTL	Data 0	Data 1	o o o	Data 18	Data 19
Value	0x6n	ASCII	ASCII		ASCII	ASCII

A great deal can be ascertained from the information returned. The following shows how this string is encoded for Cashflow-SC products:

Product	Cassette Size	Bill Path Width	Interface Option	Options	Description
SC					Cashflow-SC
	<none>				600 note cassette
	L				1200 note cassette
	M				900 note cassette
		66			66mm bill path
		83			83mm bill path (International)
		85			85mm bill path (UK)
			01		Standard RS-485 Interface
			04		Optically Isolated EBDS
			07		Standard RS-232 Interface
			21		Retail RS-485 Interface
			27		Retail RS-232 Interface
			28		Retail USB Interface
				<none>	None
				-R	Retail Kit (Hood, Slam Latch, and Retail specific firmware)
				-RB	The same as –R with the addition of a Bunch Note Feeder.
				-RE	The same as –R with the ability to handle notes up to 177mm long.
				-RL	The same as –R with the addition of a Cassette Lock.
				-RLB	The same as –R with the addition of a Cassette Lock and a Bunch Note Feeder.

Note: if the Recognition Unit (Head) has been replaced, it is possible that the acceptor type string will be incorrect if the replacement unit was not of exactly the same type.

7.4.6 Query Acceptor Serial Number

CFSC

This command is used to return the serial number of the Recognition Unit (Head). The query acceptor serial number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x05

The data returned by the device takes the form of a ASCII string that is either 20 bytes long or is terminated by a non-printable character. The reply packet is shown below (STX, Length=0x19, ETX, and CHK omitted):

Byte	3	4	5		22	23
Name	CTL	Data 0	Data 1	o o o	Data 18	Data 19
Value	0x6n	ASCII	ASCII		ASCII	ASCII

Some useful data can be ascertained from the serial number returned. The following shows how this string is encoded for Cashflow-SC products:

Week of Year	Last Digit of Year	Location	Configuration Code	Sequential Count	Description
00..51					The number of the week when the unit was manufactured.
	0..9				The number of the year when the unit was manufactured.
		0..9			The site where the unit was manufactured.
			00..99		The configuration code (build standard)
				00000 through 99999	A sequential number for units made that week.

7.4.7 Query Acceptor Boot Part Number

CFSC

This command is used to return the software part number of the boot component of the device firmware. The query acceptor boot part number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x06

The data returned by the device takes the form of a ASCII string that is 9 bytes long. The reply packet is shown below (STX, Length=0x0E, ETX, and CHK omitted):

Byte	**3**	**4**	**5**		**11**	**12**
Name	CTL	Data 0	Data 1	o o o	Data 7	Data 8
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The part number is laid out according to the following rules:

Prefix	**Project Number**	**Check Digit**	**Version**	**Description**
28				General Software Prefix
	000 .. 999			Sequential part number
		0..9		Check digit.
			000..999	Formatted as V1.23

Note: The boot software component is factory installed and should not be changed, adjusted or used as a trigger/input for any host system action, function or mode.

7.4.8 Query Acceptor Application Part Number

CFSC

This command is used to return the software part number of the file containing the application component of the device firmware. The query acceptor application part number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x07

The data returned by the device takes the form of a ASCII string that is 9 bytes long. The reply packet is shown below (STX, Length=0x0E, ETX, and CHK omitted):

Byte	3	4	5		11	12
Name	CTL	Data 0	Data 1	o o o	Data 7	Data 8
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The part number is laid out according to the following rules:

Prefix	Project Number	Check Digit	Version	Description
28				General Software Prefix
	000 .. 999			Sequential part number.
		0..9		Check digit.
			000..999	Formatted as V1.23

7.4.9 Query Acceptor Variant Name

CFSC

This command is used to return the name of the variant component of the firmware. The variant software determines which bank notes are accepted by the unit and the name of the variant, identifies the country of origin of those bank notes. The query acceptor variant name command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x08

The data returned by the device takes the form of a ASCII string that is either 32 bytes long or is terminated by a non-printable character. The reply packet is shown below (STX, Length=0x25, ETX, and CHK omitted):

Byte	3	4	5		34	35
Name	CTL	Data 0	Data 1	o o o	Data 30	Data 31
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The names of the currencies supported are represented as three character ISO codes. If more than one currency is supported, they are separated by underscore "_" characters. For example "USD_CAD" would signify a mixed U.S.A./Canadian bill set. For further information on currency descriptors, please see http://en.wikipedia.org/wiki/ISO_4217.

7.4.10 Query Acceptor Variant Part Number

CFSC

This command is used to return the software part number of the file containing the variant component of the device firmware. The query acceptor variant part number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x09

The data returned by the device takes the form of a ASCII string that is 9 bytes long. The reply packet is shown below (STX, Length=0x0E, ETX, and CHK omitted):

Byte	3	4	5		11	12
Name	CTL	Data 0	Data 1	o o o	Data 7	Data 8
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The part number is laid out according to the following rules:

Prefix	Project Number	Check Digit	Version	Description
28				Combined file Software Prefix
49				Variant Software Prefix
	000 .. 999			Sequential part number
		0..9		Check digit.
			000..999	Formatted as V1.23

Combined Files and Application/Variant Part Numbers: It is possible to load a unit with both an application and a variant at the same time, with one file called a combined file. In this case, retail units shall return the part number of the combined file for both the application component (see section 7.4.8) and the variant component. This can be deduced by the fact that the variant and the application will have the same part number. If software components are installed normally, the part numbers of each individual component is returned for the application and the variant.

Non-Retail

For some versions of the non-retail Cashflow-SC product, the individual software components' part numbers are returned regardless of how they were loaded and the combined file's part number is never returned.

7.4.11 Query Acceptor Audit Life Time Totals

CFSC

This command is used to return life time audit data kept on certain key operating data. The query acceptor audit life time totals command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0A

This data is formatted as an array of 32-bit integers where each integer is nibble encoded as eight extended data bytes. The data takes the following form (STX, Length=variable, ETX, and CHK omitted):

Byte	3	4	5		10	11
Name	CTL	Field1.0	Field1.1	o o o	Field1.6	Field1.7
Value	0x6n	Data	Data		Data	Data

Similar for Field 2

Similar for Field 3

Similar for Field N-1

FieldN.0	FieldN.1	o o o	FieldN.6	FieldN.7
Data	Data		Data	Data

Where the array of fields are mapped as:

Field Number	Size in Bytes	Data Width	Description
1	8	32	Data Map ID. The revision of data reporting in this command, Query Acceptor Audit QP Measures and Query Acceptor Audit Performance Measures. 1 – The initial revision.
2	8	32	Total Operating Hours
3	8	32	Total Motor Starts
4	8	32	Total Documents Reached Escrow Position
5	8	32	Total Notes Recognized
6	8	32	Total Notes Validated

The array of fields may be extracted as shown below (using “C” style array indexing starting at 0 rather than 1).

```
num_fields = ((Reply[1] – 5) / 8;
for (i = 0; i < num_fields; i++)
{
   Fields[i] = ((Reply[8*i +  4] & 0x0F) << 28) +
               ((Reply[8*i +  5] & 0x0F) << 24) +
               ((Reply[8*i +  6] & 0x0F) << 20) +
               ((Reply[8*i +  7] & 0x0F) << 16) +
               ((Reply[8*i +  8] & 0x0F) << 12) +
               ((Reply[8*i +  9] & 0x0F) <<  8) +
               ((Reply[8*i + 10] & 0x0F) <<  4) +
               ((Reply[8*i + 11] & 0x0F)      );
}
```

Note: In theory, up to 15 data values may be returned by this command.

7.4.12 Query Acceptor Audit QP Measures

CFSC

This command is used to return "QP" audit data kept on the general rate of bill acceptance. The query acceptor audit QP measures command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0B

This data is formatted as an array of 16-bit integers where each integer is nibble encoded as four extended data bytes. The returned data takes the following form (STX, Length=Variable, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**
Name	CTL	Field1.0	Field1.1	Field1.2	Field1.3
Value	0x6n	Data	Data	Data	Data

Similar for Field 1

o
o
o

Similar for Field N-1

FieldN.0	FieldN.1	FieldN.2	FieldN.3
Data	Data	Data	Data

Where the array of fields are mapped as:

Field Number	Size in Bytes	Data Width	Description
0	4	16	Last 100 bills acceptance rate.
1	4	16	Total Motor Starts.
2	4	16	Total Documents Stacked.
3	4	16	Total Documents Reached Escrow Position
4	4	16	Total Documents Passed Recognition

Field Number	Size in Bytes	Data Width	Description
5	4	16	Total Documents Passed Validation
6	4	16	Total Recognition Rejections
7	4	16	Total Security Rejections
8	4	16	Total Orientation Disabled Rejections
9	4	16	Total Document Disabled Rejections
10	4	16	Total Fast Feed Error Rejections
11	4	16	Total Documents Inserted While Disabled
12	4	16	Total Host Return Document Rejections
13	4	16	Total Barcodes Decoded

The array of fields may be extracted as shown below (using “C” style array indexing starting at 0 rather than 1).

```
num_fields = (Reply[1] – 5) / 4;
for (i = 0; i < num_fields; i++)
{
   Fields[i] = ((Reply[4*i + 4] & 0x0F) << 12) +
               ((Reply[4*i + 5] & 0x0F) <<  8) +
               ((Reply[4*i + 6] & 0x0F) <<  4) +
               ((Reply[4*i + 7] & 0x0F)      );
}
```

Note: In theory, up to 30 data values may be returned by this command.

7.4.13 Query Acceptor Audit Performance Measures

CFSC

This command is used to return audit data kept on the basic performance of the bill acceptor mechanism. The query acceptor audit performance measures command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0C

This data is formatted as an array of 16-bit integers where each integer is nibble encoded as four extended data bytes. The returned data takes the following form (STX, Length=Variable, ETX, and CHK omitted):

Byte	3	4	5	6	7
Name	CTL	Field1.0	Field1.1	Field1.2	Field1.3
Value	0x6n	Data	Data	Data	Data

Similar for Field 2

o
o
o

Similar for Field N-1

FieldN.0	FieldN.1	FieldN.2	FieldN.3
Data	Data	Data	Data

Where the array of fields are mapped as:

Field Number	Size in Bytes	Data Width	Description
0	4	16	Total Cross Channel 0 Rejects
1	4	16	Total Cross Channel 1 Rejects
2	4	16	Total of All Types of Jams
3	4	16	Total Jam Recovery Efforts
4	4	16	Total Reject Attempts with Jam
5	4	16	Total Stacker Jams
6	4	16	Total number of Jams without Recovery Enabled
7	4	16	Total number of Out of Service conditions
8	4	16	Total number of Out of Order conditions
9	4	16	Total number of Operating Hours
10	4	16	Total number of documents greater than maximum allowable length
11	4	16	Total number of documents less than minimum allowable length
12	4	16	Total number of documents that failed to reach escrow position
13	4	16	Total number of Calibrations
14	4	16	Total number of Resets
15	4	16	Total number of Flash Download attempts
16	4	16	Total number of Cassette Full conditions
17	4	16	Total number of Cassette Removed conditions

The array of fields may be extracted as shown below (using “C” style array indexing starting at 0 rather than 1).

```
num_fields = (Reply[1] – 5) / 4;
for (i = 0; i < num_fields; i++)
{
   Fields[i] = ((Reply[4*i + 4] & 0x0F) << 12) +
               ((Reply[4*i + 5] & 0x0F) <<  8) +
               ((Reply[4*i + 6] & 0x0F) <<  4) +
               ((Reply[4*i + 7] & 0x0F)      );
}
```

Note: In theory, up to 30 data values may be returned by this command.

7.4.14 Query Device Capabilities

This command is used to query the device capabilities. In general, this command should only be sent to devices that have indicated support by setting the DeviceCaps bit in a poll reply (see section 7.1.2). However this may not always be possible. Since some hosts do not tolerate the setting of the DeviceCaps bit, an alternate method must be found for determining device capabilities. In this method, Query Software CRC and a Query Device Capabilities commands are sent to the device. If the Data 0 ... Data 5 reply bytes are the same, then Query Device Capabilities is not supported. If the results are different, then the data returned by Query Device Capabilities may be processed. The Query Device Capabilities CRC command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0D

The reply from the device takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	Cap0	Cap1	Cap2	Cap3	Cap4	Cap5

The Cap bytes are used to represent various device capabilities. These are specified in the following table:

Cap	Bit	Capability
0	0	1 = Extended PUP mode is supported.
	1	1 = Extended orientation handling is supported.
	2	1 = QueryAcceptorApplicationID and QueryAcceptorVariantID are supported.
	3	1 = QueryBNFStatus is supported
	4	1 = Test Documents are supported.
	5	1 = Set Bezel is supported.
	6	Reserved, 0
1	All	Reserved, 0
2	All	Reserved, 0
3	All	Reserved, 0
4	All	Reserved, 0
5	All	Reserved, 0

Note that reserved fields may be used to describe new capabilities at any time. Host code should not interrogate such reserved fields.

7.4.15 Query Acceptor Application ID

CFSC

This command is used to return the software part number of the actual application component of the device firmware. The query acceptor application part number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0E

The data returned by the device takes the form of a ASCII string that is 9 bytes long. The reply packet is shown below (STX, Length=0x0E, ETX, and CHK omitted):

Byte	**3**	**4**	**5**		**11**	**12**
Name	CTL	Data 0	Data 1	o o o	Data 7	Data 8
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The part number is laid out according to the following rules:

Prefix	Project Number	Check Digit	Version	Description
28				General Software Prefix
	000 .. 999			Sequential part number
		0..9		Check digit.
			000..999	Formatted as V1.23

7.4.16 Query Acceptor Variant ID

CFSC

This command is used to return the software part number of the actual variant component of the device firmware. The query acceptor variant part number command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x0F

The data returned by the device takes the form of a ASCII string that is 9 bytes long. The reply packet is shown below (STX, Length=0x0E, ETX, and CHK omitted):

Byte	3	4	5		11	12
Name	CTL	Data 0	Data 1	o o o	Data 7	Data 8
Value	0x6n	ASCII	ASCII		ASCII	ASCII

The part number is laid out according to the following rules:

Prefix	Project Number	Check Digit	Version	Description
28				Combined file Software Prefix
49				Variant Software Prefix
	000 .. 999			Sequential part number
		0..9		Check digit.
			000..999	Formatted as V1.23

7.4.17 Query BNF Status

CFSC

This command is used to determine the status of the Bunch Note Feeder attachment. This command may only be called if the QueryBNFStatus bit is set in the device capability map (see section 7.4.14). The command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	00	00	0x10

The reply takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	Present	Status	00	00	00	00

Where:

Name	**Value**	**Meaning**
Present	0	A BNF is not detected.
	1	A BNF is detected.
	Other	Reserved
Status	0	OK
	1	An error has been detected
	Other	Reserved

7.4.18 Set Bezel

CFSC

This command is used to override the default bezel configuration of the bill acceptor bezel. This command may only be called if the SetBezel bit is set in the device capability map (see section 7.4.14). The command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**
Name	CTL	Data A	Data B	Command
Value	0x6n	Bezel	00	0x11

Where the bezel value is:

Bezel Code	Bezel
0x00	Standard Bezel
0x01	Platform Bezel
0x02	Enhanced Diagnostic Bezel

The reply from the device takes the form (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x6n	0x00	0x00	0x00	0x00	0x00	0x00

Note: Under some conditions, the bill acceptor will perform a soft reset after this command. In current models, this occurs when changing from a platform bezel to a different bezel. The bezel setting is currently stored in the interface card microcontroller.

7.4.19 Acceptor Soft Reset

Gen2D **S3K** **CFSC**

This command is used to reset the bill acceptor. There is not necessarily a reply to this command, but some data may be sent by the device. The host system should ignore all data sent by the device for at least one second. Further, the device may take as much as fifteen seconds to return to normal operation after being reset and the host should poll, once per second, for at least fifteen seconds until the device replies. The acceptor soft reset command takes the form (STX, Length=0x08, ETX, and CHK omitted):

Byte	3	4	5	6
Name	CTL	Data A	Data B	Command
Value	0x6n	0x7F	0x7F	0x7F

CAUTION: The intent of this command is to permit a host system to establish an initial condition when the software is launched. The use of the Acceptor Soft Reset command to clear error conditions (such as jammed, failure, or cashbox full) is not recommended as this may cause a problem to become more severe.

7.5 The Extended Commands

The extended commands utilize message type 7 to provide functionality outside that provided by the standard omnibus commands. The use of message type 7 is complicated by the fact that it is used for special host commands, replies to special host commands and special replies to standard commands. To facilitate these uses, the following table is provided:

Type	Sub Type	Length	Host	Device
7x	0x01	0x28	Not used	A Bar Code reply to the standard command.
	0x02	0x0A	Query Extended Note Specification command	Not used
		0x1E	Not used	The reply to the Query Extended Note Specification command
				An Expanded Note reply to the standard command.
	0x03	0x0C	Not used.	A reply to the Set Expanded Note Inhibits for some bill acceptors. See section 7.5.2 for details.
		0x11	Set Expanded Note Inhibits command.	Not used.
	0x04	0x0B	Set Escrow Timeout command.	Not used.
		0x0C	Not used.	The reply to the Set Escrow Timeout command.
		0x12	Not used.	An Expanded Coupon reply to the standard command.
	0x05	0x0C	Not used.	The reply to the Set Asset Number command.
		0x19	Set Asset Number command.	Not used.
	0x06	*Do Not Use / Reserved*		
	0x07	0x0C	Not used.	The reply to the Set Extended PUP command.
		0x0E	Set Extended PUP command.	Not used.

Note: Lengths other than those listed in the table represent invalid packets.

7.5.1 Query Expanded Note Specification

CFSC

When expanded bank note processing is enabled, this command is used to retrieve the specification of a note. The format of this command is shown below (STX, Length=0x0A, ETX, and CHK omitted):

Byte	3	4	5	6	7	8
Name	CTL	Subtype	Data 0	Data 1	Data 2	Index
Value	0x7n	0x02	nn	nn	nn	nn

Where Data 0 through Data 2 are defined the same as the similar values in the standard omnibus command (see section 7.1.1) but do note that the position of these bytes is shifted one position due to the presence of a subtype byte. The Index value is the index of the note to be queried. This index ranges from 1 though to 1 beyond the last defined note. The data returned by the bill acceptor is formatted as (STX, Length=0x1E, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4
Value	0x7n	0x02	nn	nn	nn	nn	nn

10	11	12	o o o	28
Data 5	Ext Data 0	Ext Data 1		Ext Data 17
nn	nn	nn		nn

The extended bill data is described below:

Field	Byte Offset	Field Description	Sample Value (2000 Yen Note)
Index	0	If this is a valid bill, then the value of this byte should match the value of the Index in the command. If this value is zero, then the host has reached the end of the bill table and should stop iterating.	2
ISO Code	1..3	A three character ASCII currency code. See ISO 4217 for details	“JPY”
Base Value	4..6	A three character ASCII coded decimal value	“002”
Sign	7	An ASCII coded sign value for the Exponent. This field is either a “+” or a “-“	“+”
Exponent	8..9	ASCII coded decimal value for the power of ten that the base is to either be multiplied by (if Sign is “+”) or divided by (if Sign is “-“)	“03”
Orientation	10	Not used. Always 0x00.	0x00
Type	11	An ASCII letter that documents the note type. This corresponds to the data in the variant identity card.	“A”
Series	12	An ASCII letter that documents the note series. This corresponds to the data in the variant identity card.	“A”

Field	Byte Offset	Field Description	Sample Value (2000 Yen Note)
Compatibility	13	An ASCII letter that documents the revision of the recognition core used. This corresponds to the data in the variant identity card.	"B"
Version	14	An ASCII letter that documents the version of the note's recognition criteria. This corresponds to the data in the variant identity card.	"A"
Reserved	15..17	3 bytes reserved for future use.	N/A

7.5.2 Set Expanded Note Inhibits

CFSC

This command is used to control the acceptance of bank notes on a note type basis. This command is formatted as follows STX, Length=0x11, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**
Name	CTL	Sub Type	Data 0	Data 1	Data 2
Value	0x7n	0x03	nn	nn	nn

8	**9**		**15**
Enable 1	Enable 2	o o o	Enable 8
nn	nn		nn

Where Data 0 through Data 2 are defined the same as the similar values in the standard omnibus command (see section 7.1.1) but do note that the position of these bytes is shifted one position due to the presence of a subtype byte. The Enable data is used to enable bills by index. This is shown below:

	Bit 6	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Enable 1	Bill 7	Bill 6	Bill 5	Bill 4	Bill 3	Bill 2	Bill 1
Enable 2	Bill 14	Bill 13	Bill 12	Bill 11	Bill 10	Bill 9	Bill 8
Enable 3	Bill 21	Bill 20	Bill 19	Bill 18	Bill 17	Bill 16	Bill 15
Enable 4	Bill 28	Bill 27	Bill 26	Bill 25	Bill 24	Bill 23	Bill 22
Enable 5	Bill 35	Bill 34	Bill 33	Bill 32	Bill 31	Bill 30	Bill 29
Enable 6	Bill 42	Bill 41	Bill 40	Bill 39	Bill 38	Bill 37	Bill 36
Enable 7	Bill 49	Bill 48	Bill 47	Bill 46	Bill 45	Bill 44	Bill 43
Enable 8	-	-	-	-	-	-	Bill 50

Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. The reply contains no extended data (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x2n	nn	nn	nn	nn	nn	nn

In some bill acceptors, an alternate reply is given. This reply also contains no extended data (STX, Length=0x0C, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9	10
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x7n	0x03	nn	nn	nn	nn	nn	nn

7.5.3 Set Escrow Timeout

CFSC **Non-Retail**

This command is used to set the escrow timeout of the bill acceptor. This command is formatted as follows (STX, Length=0x0B, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Notes	Barcode
Value	0x7n	0x04	nn	nn	nn	nn	nn

Where Data 0 through Data 2 are defined the same as the similar values in the standard omnibus command (see section 7.1.1) but do note that the position of these bytes is shifted one position due to the presence of a subtype byte. The Notes and Barcode fields set the timeout for bank notes and barcodes respectively. This is a value from 1 through 127 seconds, or zero to disable the timeout. By default, both timeouts are disabled. The reply contains no extended data (STX, Length=0x0C, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9	10
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x7n	0x04	nn	nn	nn	nn	nn	nn

Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2.

7.5.4 Set Asset Number

CFSC **Non-Retail**

This command is used to write a string into the asset number of the bill acceptor and the optional non-volatile memory tag installed in the cash box. This allows the cash box to be linked back to a specific bill acceptor at a later time when the tag is read. This command is shown below (STX, Length=0x19, ETX, and CHK omitted).

Byte	3	4	5	6	7
Name	CTL	Sub Type	Data 0	Data 1	Data 2
Value	0x7n	0x05	nn	nn	nn

8	9		23
Asset 0	Asset 1	o o o	Asset 16
nn	nn		nn

Where Data 0 through Data 2 are defined the same as the similar values in the standard omnibus command (see section 7.1.1) but do note that the position of these bytes is shifted one position due to the presence of a subtype byte. The asset bytes contain an asset string (number) that is programmed into the unit.

In the reply, Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. The reply contains no extended data (STX, Length=0x0C, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9	10
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x7n	0x05	nn	nn	nn	nn	nn	nn

Note: this command is NOT supported in the current retail code base.

7.5.5 Query Value Table

CFSC **Deprecated** **Non-Retail**

This command is specific to one particular customer and should not be used. It is not documented here.

Note: this command is NOT supported in the current retail code base.

7.5.6 Set Extended PUP Mode

CFSC

This command is used to set up extended PUP mode processing in those bill acceptors that support it. This may be determined by examining the Device Capability Map (see section 7.4.11 for details). This command is formatted as follows STX, Length=0x0E, ETX, and CHK omitted):

Byte	**3**	**4**	**5**	**6**	**7**
Name	CTL	Sub Type	Data 0	Data 1	Data 2
Value	0x7n	0x07	nn	nn	nn

8	**9**	**10**	**11**	**12**
Mode	Pre_Escrow	At_Escrow	Post_Escrow	Pre_Stack
nn	nn	nn	nn	nn

Where Data 0 through Data 2 are defined the same as the similar values in the standard omnibus command (see section 7.1.1) but do note that the position of these bytes is shifted one position due to the presence of a subtype byte. The Mode value is interpreted as follows:

Mode	Description
"A"	Use PUP-A mode. Pre_Escow, At_Escrow, Post_Escrow and Pre_Stack are ignored.
"B"	Use PUP-B mode. Pre_Escow, At_Escrow, Post_Escrow and Pre_Stack are ignored.
"C"	Use PUP-C mode. Pre_Escow, At_Escrow, Post_Escrow and Pre_Stack are ignored.
"E"	Use PUP-E mode. In this mode, the normal PUP bits are ignored and Pre_Escow, At_Escrow, Post_Escrow and Pre_Stack specify the PUP behaviour. See below.
Other	Use PUP-A mode. Pre_Escow, At_Escrow, Post_Escrow and Pre_Stack are ignored.

PUP processing is controlled by where the bill was when the power failed. The following positions are defined for the purposes of power up handling.

Bill Position	Description
Pre_Escrow	The bill is in the process of being drawn into the unit. A determination of the type of the bill has not yet been made.
At_Escrow	The bill is at escrow and has been verfied.
Post_Escrow	The command to stack the note was given, but the bill may still be returned.
Pre_Stack	The command to stack the note was given, but the bill may not be returned.

The control of the power-up process is shown in the table below:

Value	Pre Escrow	At Escrow	Post Escrow	Pre Stack	Action
0	✓	✓	✓	✓	Go out of service.
1	✓	✓	✓	✓	Stack the bill without credit.
2	✓	✓	✓	-	Return the bill.
3	-	✓	✓	✓	Stack the bill with credit.
4	-	✓	-	-	Erase credit and wait for the host to decide.
5	-	✓	-	-	Preserve credit and wait for the host to decide.

In the reply, Data 0 through Data 5 are interpreted just as in the standard response detailed in section 7.1.2. The reply contains no extended data (STX, Length=0x0C, ETX, and CHK omitted):

Byte	3	4	5	6	7	8	9	10
Name	CTL	Sub Type	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5
Value	0x7n	0x07	nn	nn	nn	nn	nn	nn

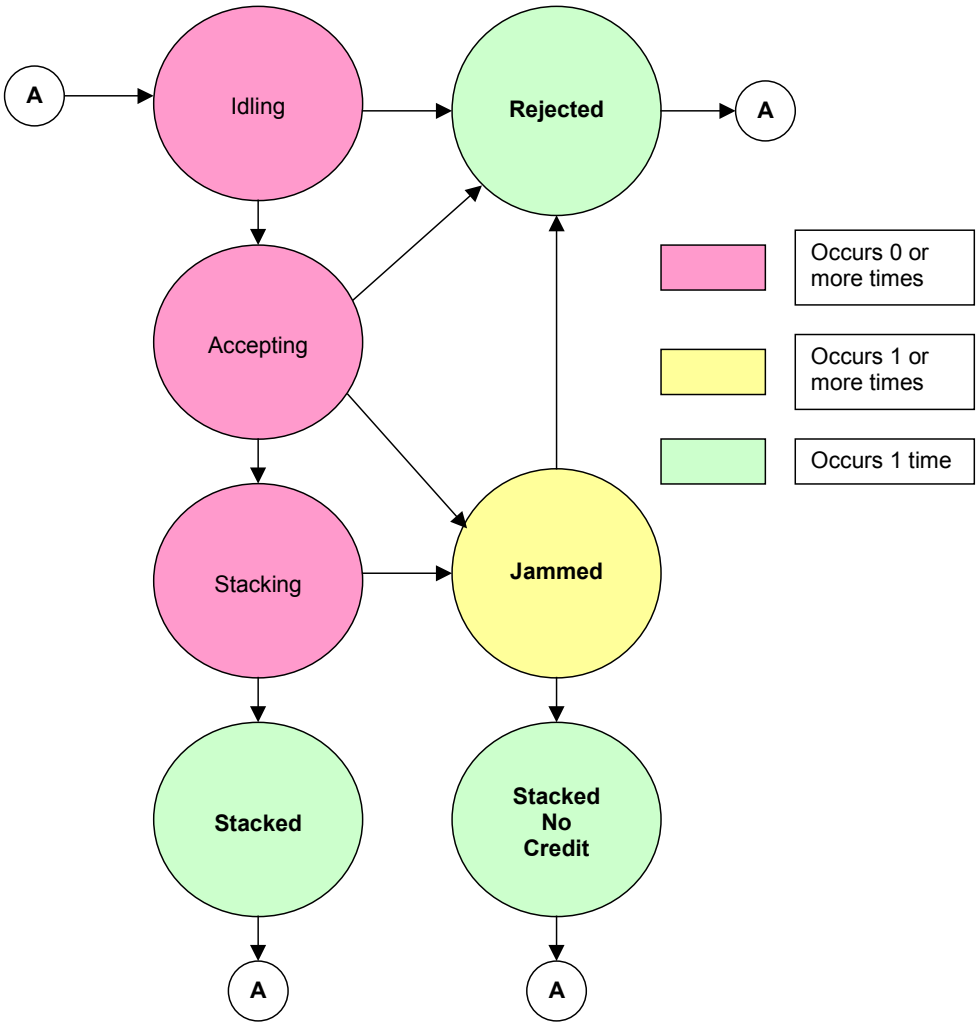
7.6 Processing States

This section shall describe the processing states associated with the handling of bank notes and documents in two basic modes of operation.

7.6.1 Processing States in Non-escrow Mode

Deprecated

The following illustrates the processing states normally encountered while handling a bill in non-escrow mode:

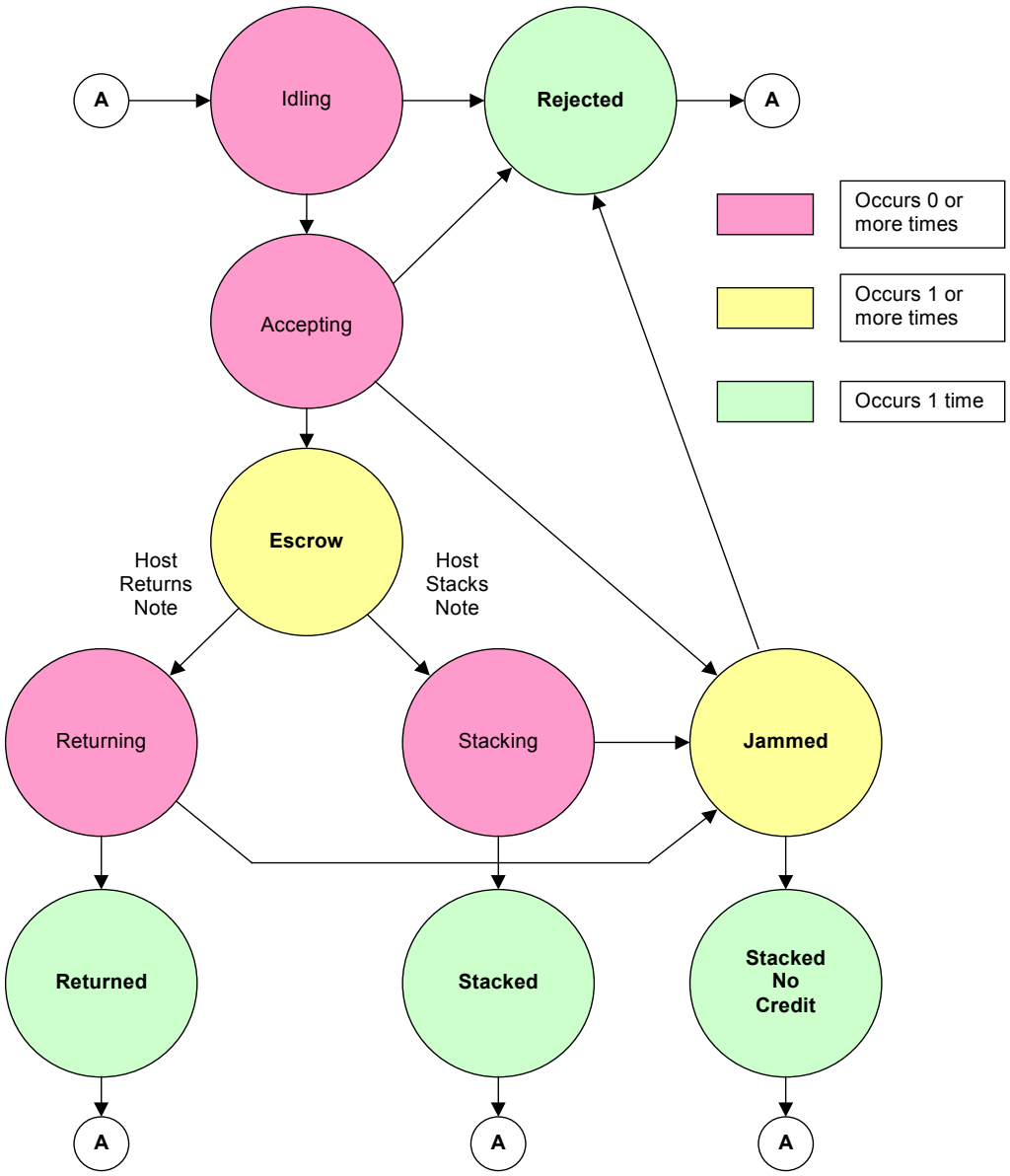


Note that the value of notes is only transmitted to the host when the note is stacked. In case of a problem stacking the note or a jam, there is no way to determine the value of notes accepted. This can be contrasted with Escrow mode (illustrated in the next section) where the host is able to record the value of notes at escrow.

7.6.2 Processing States in Escrow Mode

Recommended

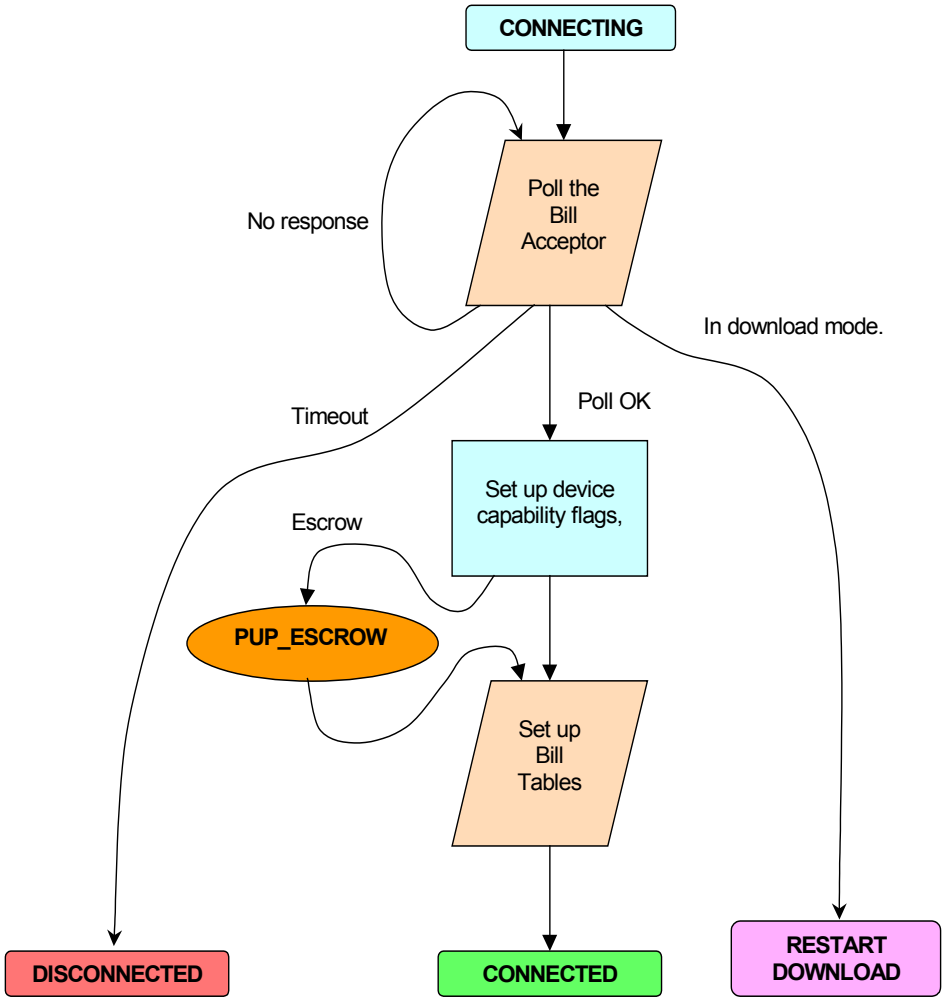
The following illustrates the processing states normally encountered while handling a bill in escrow mode:



8. The Application Layer (for Bill Acceptor)

8.1 Application Startup Tasks

When the application first begins communicating with the bill acceptor, it starts off polling the device. Depending on the response of the device, appropriate action is taken to set up the bill acceptor. This is outlined below:



There are four possible outcomes:

CONNECTED	The acceptor is ready for further commands.
PUP_ESCROW	An intermediate state in which the host must decide to accept or reject a note in escrow at the time the bill acceptor powered up.
RESTART DOWNLOAD	The acceptor needs to have an application loaded into it.
DISCONNECTED	The attempt to connect did not succeed

8.2 Application Currency Handling

8.2.1 Handling Money in Terse Mode:

While expanded mode is preferred, it is not supported across all production platforms. In those cases, the terse mode must be used. One issue that is raised by terse mode is that bank notes are not identified as to their value, but are merely given an identifier from 1 through 7. The following table shows how, with the help of the model number byte defined in section 7.1.2 it is usually possible to determine the value of currency:

Country	USA		Argentina	Australia		Brazil	Canada	Europe	Mexico	Russia	Unknown
ISO Code	USD		ARS	AUD		BRL	CAD	EUR	MXP	RUR	***
Model Numbers	20, 31, "J", "X"	"P"	"G"	"A"	15	"W"	"C"	"D"	"M"	"B"	Other
Index											
1	$1	$1	-	$5	-	1R	-	€5	20P	10R	$1
2	$2	$2	2P	$10	-	2R	-	€10	50P	50R	$2
3	$5	$5	5P	$20	$5	5R	$5	-	-	100R	$5
4	$10	$10	10P	-	$10	10R	$10	-	-	500R	$10
5	$20	$20	20P	-	$20	20R	$20	-	-	-	$20
6	$50	-	50P	-	$50	50R	$50	-	-	-	$50
7	$100	-	100P	-	$100	100R	$100	-	-	-	$100

Note that for most cases, the US table of $1 through $100 may be used. Those entries above marked with (US) may safely use the US table. When confronted with an unknown model, this table should be used by default. In other cases, an alternate bill table will need to substituted for the default table when the required model number is detected.

8.2.2 Handling Money in Expanded Mode:

If the model number of the EBDS unit is "T" or "U" then the device supports expanded note reporting. When ever possible, expanded note reporting should be used to enhance code portability and reliability. Expanded note reporting allows applications to be used in various locales without having to be rewritten. It allows for changes in the currency and provides for much finer control over the type and orientation of notes accepted.

The amount of a expanded note report may be extracted with the following snippet of "C" code.

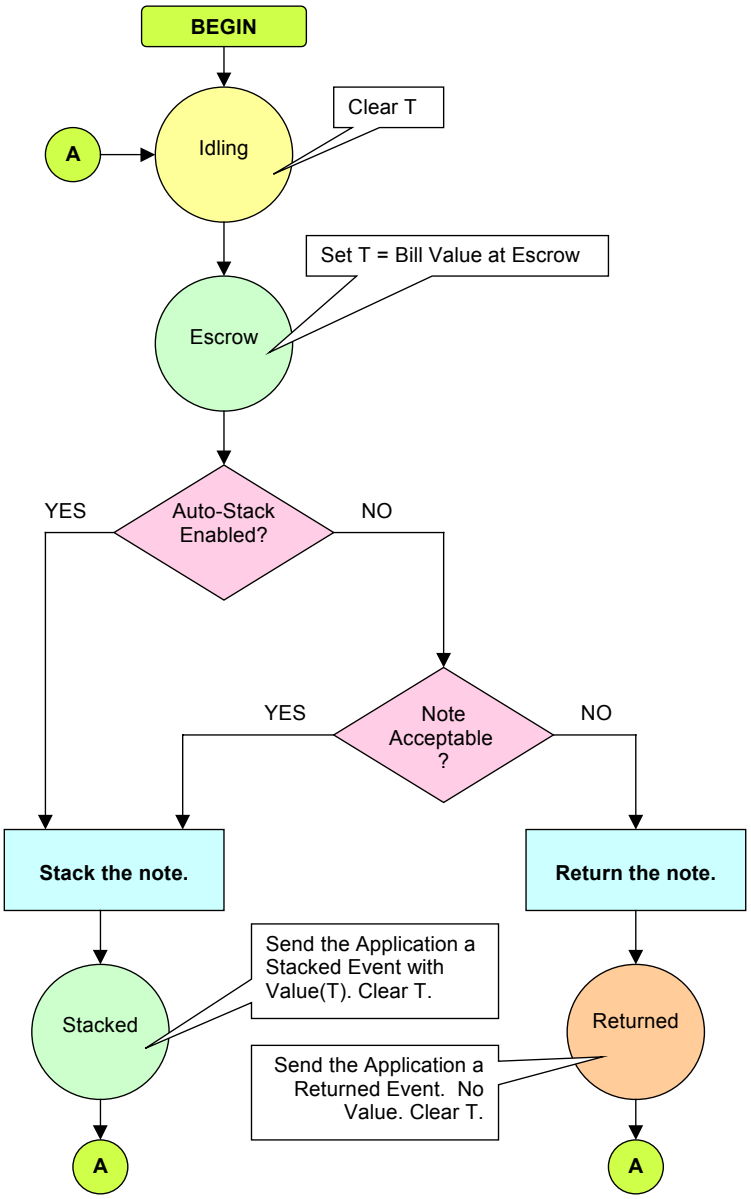
```
unsigned long N,D;    // Numerator and Denominator.
int E;                // Multiplier Exponent.
double Amount;        // Amount of Currency.
int i;

N = (ExtData[4] – '0') * 100 +
    (ExtData[5] – '0') * 10 +
    (ExtData[6] – '0');
```

```
E = (ExtData[8] – '0') * 10 +
    (ExtData[9] – '0');

if (ExtData[7] == '+')
{
      for (i = 0; i < E; i++)
            N *= 10;

      Amount = (Double)N;
}
else if (ExtData[7] == '-')
{
      D = 1;

      for (i = 0; i < E; i++)
            D *= 10;

      Amount = (Double)N/(Double)D;
}
else
{
      Amount = (Double)0.0;
}
```

8.2.3 Recommended money handling flowchart:

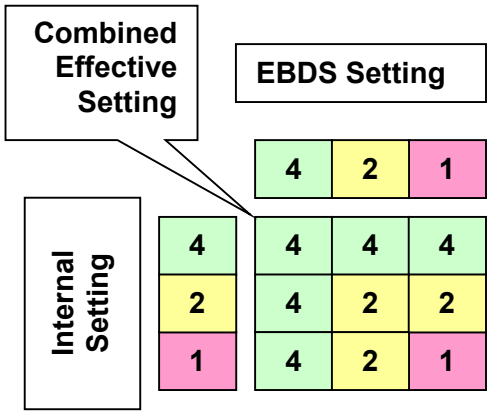
The following diagram depicts the recommended method of handling money. There are two main things to note:

- The deprecated non-escrow mode is replaced by the concept of the "auto-stack" property. If the application is designed to simply accept bank notes as they come in, the auto-stack property should be true, so that stacking occurs automatically. If the application needs more control over which notes are accepted, then the auto-stack property should be false. When auto-stack is false, the application will receive an "Escrow" event and can then use the StackEscrowNote() or ReturnEscrowNote() methods to accept or reject the note.
- The value of the bill is recorded while it is at escrow. It is this recorded value that is used when the bill is stacked and the application receives a "Stacked" event. This handling of the bill value is needed to handle the cases where a bill is stacked with no credit. Since the value is retained from escrow, errors in credit and cash box counts are prevented.



8.2.4 Controlling the orientation of accepted bills:

In some applications, it is desired to control the orientation of accepted bills. To accomplish this, the orientation control bits of the omnibus command may be used to set the orientation (see section 7.1.1). This method is of limited control however. The reason is that the bill acceptor also has an internal orientation control. Both of these settings determine the actual orientation setting used according to the following table:



It can be seen that the most "liberal" possible setting is used by the acceptor. In this situation, the only way for the host to have complete control over the orientation of accepted bank notes is to have the units internal settings configured for 1-way acceptance. Conversely, if it desired to have control over the orientation of notes by means of the bill acceptor's internal settings, the EBDS interface will have to be set to 1-way acceptance.

8.2.5 Improved control of the orientation in expanded mode:

For those devices that support expanded note reporting, it is possible for the host to attain a greater level of control over the orientation of accepted notes. Quite simply, when notes arrive at escrow, the host can examine the orientation field (see sections 7.1.4 and 7.4.14) in the expanded note data and simply return any notes that do correspond to the desired orientation. In this mode, the host system has the final say as to the suitability of the note.

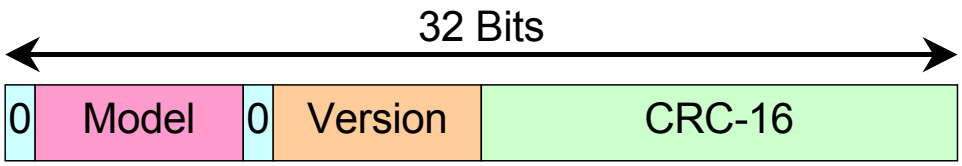
8.3 Determining the Firmware Version

When a host system connects to a bill acceptor, it is often useful to ascertain the version and type of software installed in the unit. If the host can determine this, it is capable then of making decisions about the firmware like performing a upgrade to the desired firmware or signalling an error that the software in the unit does not match the expected software.

8.3.1 Firmware Version in Classic EBDS:

In classic EBDS, it is not generally possible to determine the exact firmware stored in the flash memory of a device. It is however possible to build a unique "signature" of the code and to use this as a form of identification. This is accomplished by means of the Model Number byte, the Code Revision byte and the Flash CRC. Together these form a 32 bit code signature that should be unique. If this signature does not match the expected signature, the corrective action can be taken in the form of uploading the appropriate firmware or signaling an error condition.

This 32 bit code signature could be laid out as follows:



The integer value created could then be used to key into a database or some similar scheme.

8.3.2 Firmware Version in Extended EBDS:

In systems that support the extended command set, it is possible to directly query the part number of the application and variant installed in the unit. This returns two strings that permit the host system to directly determine if the software manifest of the bill acceptor is correct. As above, corrective actions can include uploading the appropriate firmware or signaling an error condition.

8.4 Handling Acceptor Exceptions

Some acceptor exceptions bear further examination. This section will review suggested corrective actions to be taken when unusual status values are returned.

8.4.1 Bill Acceptor does not respond to a poll:

It is not unusual for the bill acceptor to be busy and to “miss” a poll. Under these conditions the host should retry at the normal poll rate with the same ACK value (see section 4.5). If after ten retries, there has been no response it should toggle the ACK value and keep trying for ten more times.

8.4.2 Bill Acceptor does not respond for an extended period:

If the bill acceptor does not respond after thirty seconds of retrying, the host should “declare” the unit is out of service and that service intervention is required. The most likely cause of the problem is that there is a cabling or power problem with the unit or there is something wrong with the bill acceptor. The host should go out of service and request a field service call.

8.4.3 Bill Acceptor Status: Cheated

If a bill acceptor status of cheated (see section 7.1.2) is returned to the host, it means that there was a problem transporting the bill to the cashbox. This could indeed be the result of fraudulent manipulation of the bill or the unit. It could also be as harmless as an old bank note getting stuck. In most cases, the host should simply ignore this event. MEI does not offer a method to set / test cheat events.

8.4.4 Bill Acceptor Status: Rejected

If the bill acceptor returns a status of rejected (see section 7.1.2) it simply means that a bill was not recognized as a good bill. The host should take no action.

8.4.5 Bill Acceptor Status: Jammed

If the bill acceptor returns a status of jammed (see section 7.1.2) it means that a problem has been encountered transporting the bill (either for acceptance or rejection). When a jam is detected, anti-jam routines are run to clear the jam. If successful, the jammed condition will be cleared, otherwise, it will persist. If the host detects a jammed condition, it should go out of service for the duration of the jam.

8.4.6 Bill Acceptor Status: Stacker Full

If the bill acceptor returns a status of stacker full (see section 7.1.2) it means that no more documents may be placed into the cashbox. The host should go out of service and request a cash box swap from the attendant.

8.4.7 Bill Acceptor Status: Cashbox Removed

If the bill acceptor returns a status of cash box removed (see section 7.1.2) it means that the unit may no longer accept money. This will occur as a matter of course when the cashbox is swapped at the end of the shift. It may also occur if the cashbox is not seated correctly and normal operating vibration causes it to work loose. In this case the host should go out of service and request an attendant to examine the cashbox.

8.4.8 Bill Acceptor Status: Paused

If the bill acceptor returns a status of paused (see section 7.1.2), it means that the consumer is feeding bills too fast and needs to remove the current bill from the mouth of the bill acceptor so that it can proceed. The host should signal this to the consumer.

8.4.9 Bill Acceptor Status: Calibration in Progress

If the bill acceptor returns a status of calibration in progress (see section 7.1.2) and a calibration was not requested, it means that there is a problem with the bill acceptor and the host should go out of service and request a field service call.

8.4.10 Bill Acceptor Status: Power Up

If the bill acceptor returns a status of power up (see section 7.1.2) it means that something has caused the unit to be reset or that it has lost power and recovered. The host should proceed to perform its initialization tasks once more (see section 8.1)

8.4.11 Bill Acceptor Status: Invalid Command

If the bill acceptor returns a status of invalid command (see section 7.1.2) it means that there is a defect in the host code. Normally this error should never occur outside of development and debug phases of the system. In the field, this error should be logged as a problem with the host system programming.

8.4.12 Bill Acceptor Status: Failure

If the bill acceptor returns a status of failure (see section 7.1.2) it means that there is a serious problem with either the bill acceptor, the chassis, or the cash box. This problem requires attention and the host should go out of service and request a field service call.

8.4.13 Bill Acceptor Status: Stalled

If the bill acceptor returns a status of stalled (see section 7.1.2) it means that the unit encountered a problem transporting the bill to the cashbox and that it is now stalled, with the bill near or just at the cashbox. An attendant is required to examine the source of the problem with the bill and to reset the unit. This condition only occurs if the host enables No Push mode (see section 7.1.1)

8.4.14 Bill Acceptor Status: Flash Download

If the bill acceptor returns a status of flash download (see section 7.1.2) and this is not expected, it means that the unit is lacking application code or that a previous download attempt was interrupted or failed. the host may either attempt to download the correct firmware or it may go out of service and request a field service call.

9. MEI Point of Service Toolkit (M/POST)

EBDS protocol devices may also be accessed via the EBDS portion the MEI Point of Service Toolkit (M/POST). The M/POST provides for significant savings of time and effort for coding, debugging and deploying applications that require the acceptance and handling of cash.

9.1 M/POST for EBDS Overview:

The M/POST is based on the Properties/Methods/Events model of programming used in objected oriented systems and the description of it will be along those lines.

Notes:

- For programming systems that do not directly support properties, access functions are used. The prefix "Qry" is used for the read access function and the prefix "Set" is used for the write access function.
- For programming systems that do not support name spaces, all of the following definitions (except for common ones like BOOLEAN, STRING etc) are prefixed with "Ebds".
- When both name space and access prefixes are needed, the "Ebds" prefix appears first in the entity name. This would result in function names like "EbdsQryCapOrientationExt()".

*9.1.1 M/POST for EBDS **Acceptor** Properties:*

Name	Type	Vers	Access	When usable?
ApplicationID	STRING	1.00	R	CapApplicationID
ApplicationPN	STRING	1.00	R	CapApplicationPN
AuditLifeTimeTotals	INT32[ ]	1.00	R	CapAudit.
AuditPerformance	INT32[ ]	1.00	R	CapAudit.
AuditQP	INT32[ ]	1.00	R	CapAudit.
AutoStack	BOOLEAN	1.00	R/W	Connected.
BarCode	STRING	1.00	R	DocType == DocBarCode.
Bill	Bill	1.00	R	DocType == DocBill.
BillTypes	Bill[ ]	1.00	R	Connected.
BillTypeEnables	BOOLEAN[ ]	1.00	R/W	Connected.
BillValues	Bill[ ]	1.00	R	Connected.
BillValueEnables	BOOLEAN[ ]	1.00	R/W	Connected.
BNFStatus	BNFStatus	1.00	R	CapBNFStatus
BootPN	STRING	1.00	R	CapBootPN.
CapApplicationID	BOOLEAN	1.00	R	Connected.
CapApplicationPN	BOOLEAN	1.00	R	Connected.
CapAssetNumber	BOOLEAN	1.00	R	Connected.
CapAudit	BOOLEAN	1.00	R	Connected.
CapBarCodes	BOOLEAN	1.00	R	Connected.
CapBarCodesExt	BOOLEAN	1.00	R	Connected.
CapBNFStatus	Boolean	1.00	R	Connected.
CapBookmark	BOOLEAN	1.00	R	Connected.
CapBootPN	BOOLEAN	1.00	R	Connected.

Name	Type	Vers	Access	When usable?
CapCalibrate	BOOLEAN	1.00	R	Connected.
CapCashBoxTotal	BOOLEAN	1.00	R	Connected.
CapCouponExt	BOOLEAN	1.00	R	Connected.
CapDevicePaused	BOOLEAN	1.00	R	Connected.
CapDeviceSoftReset	BOOLEAN	1.00	R	Connected.
CapDeviceType	BOOLEAN	1.00	R	Connected.
CapDeviceResets	BOOLEAN	1.00	R	Connected.
CapDeviceSerialNumber	BOOLEAN	1.00	R	Connected.
CapEscrowTimeout	BOOLEAN	1.00	R	Connected.
CapFlashDownload	BOOLEAN	1.00	R	Connected.
CapNoPush	BOOLEAN	1.00	R	Connected.
CapOrientationExt	BOOLEAN	1.00	R	Connected.
CapPupExt	BOOLEAN	1.00	R	Connected.
CapSetBezel	BOOLEAN	1.10	R	Connected.
CapTestDoc	BOOLEAN	1.00	R	Connected.
CapVariantID	BOOLEAN	1.00	R	Connected.
CapVariantPN	BOOLEAN	1.00	R	Connected.
CashBoxAttached	BOOLEAN	1.00	R	Connected.
CashBoxFull	BOOLEAN	1.00	R	Connected.
CashBoxTotal	INT32	1.00	R	CapCashBoxTotal.
Connected	BOOLEAN	1.00	R	Always.
Coupon	Coupon	1.00	R	DocType == DOC_COUPON
DebugLog	BOOLEAN	1.00	R/W	Always.
DebugLogPath	STRING	1.00	R/W	Read: Always Write: When DebugLog is false.
DeviceBusy	BOOLEAN	1.00	R	Connected.
DeviceCRC	INT32	1.00	R	Connected.
DeviceFailure	BOOLEAN	1.00	R	Connected.
DeviceJammed	BOOLEAN	1.00	R	Connected.
DeviceModel	INT32	1.00	R	Connected.
DevicePaused	BOOLEAN	1.00	R	CapDevicePaused.
DevicePortName	STRING	1.00	R	Connected.
DevicePowerUp	PowerUp	1.00	R	Connected.
DeviceResets	INT32	1.00	R	CapDeviceResets.
DeviceRevision	INT32	1.00	R	Connected.
DeviceSerialNumber	STRING	1.00	R	CapDeviceSerialNumber.
DeviceSignature	INT32	1.00	R	Connected.
DeviceStalled	BOOLEAN	1.00	R	CapNoPush.
DeviceState	State	1.00	R	Always.
DeviceType	STRING	1.00	R	CapDeviceType
DocType	DocType	1.00	R	After DeviceState == Escrow.
EnableAcceptance	BOOLEAN	1.00	R/W	Connected.
EnableBarCodes	BOOLEAN	1.00	R/W	CapBarCodes.
EnableBookmarks	BOOLEAN	1.00	R/W	CapBookmark.
EnableCouponExt	BOOLEAN	1.00	R/W	CapCouponExt
EnableNoPush	BOOLEAN	1.00	R/W	CapNoPush.
EscrowOrientation	Orientation	1.00	R	After DeviceState == Escrow. when CapOrientationExt.

Name	Type	Vers	Access	When usable?
HighSecurity	BOOLEAN	1.00	R/W	Connected.
OrientationCtl	OrientationCtl	1.00	R/W	Connected.
OrientationCtlExt	OrientationCtl	1.00	R/W	CapOrientationExt.
VariantID	STRING	1.00	R	CapVariantID.
VariantNames	STRING[ ]	1.00	R	Connected.
VariantPN	STRING	1.00	R	CapVariantPN.
Version	STRING	1.00	R	Always

*9.1.2 M/POST for EBDS **Bill** Properties:*

Name	Type	Vers	Access	Values
Country	STRING	1.00	R	Three letter ISO code or "*".
Value	DOUBLE	1.00	R	
Type	CHAR	1.00	R	'A' .. 'Z' or '*'
Series	CHAR	1.00	R	'A' .. 'Z' or '*'
Compatibility	CHAR	1.00	R	'A' .. 'Z' or '*'
Version	CHAR	1.00	R	'A' .. 'Z' or '*'

*9.1.3 M/POST for EBDS **Coupon** Properties:*

Name	Type	Vers	Access	Values
OwnerID	INT32	1.00	R	An integer assigned to the owner of the coupon
Value	DOUBLE	1.00	R	The monetary value of the generic coupon. 0.0D == Free vend coupon 1.0D == $1 coupon 2.0D == $2 coupon 5.0D == $5 coupon

*9.1.4 M/POST for EBDS **DocType** Enumeration:*

```
DocType = {None, NoValue, Bill, Barcode, Coupon}
```

This type describes the type of document currently being processed. This may be one of:

Type	Description
None	No document is currently being processed.
NoValue	A document with no value is being processed.
Bill	A bank note is being processed.
Barcode	A bar code is being processed.
Coupon	A generic coupon is being processed.

*9.1.5 M/POST for EBDS **Orientation** Enumeration:*

```
Orientation = {RightUp, RightDown, LeftUp, LeftDown,
               Unknown}
```

This type describes the orientation of a note. It is hoped that the descriptions are self evident.

*9.1.6 M/POST for EBDS **OrientationCtl** Enumeration:*

```
OrientationCtl = {FOUR_WAY, TWO_WAY, ONE_WAY }
```

This type describes the ways that the orientation of notes may be controlled. This is shown in the table below:

Control	Right-Up	Right-Down	Left-Up	Left-Down
FOUR_WAY	✓	✓	✓	✓
TWO_WAY	✓		✓	
ONE_WAY	✓			

*9.1.7 M/POST for EBDS **PowerUp** Enumeration:*

```
PowerUp = {A, B, C, E}
```

*9.1.8 M/POST for EBDS **PupExt** Enumeration:*

```
PupExt = {PUP_RETURN, PUP_OOS, PUP_STACK_NC, PUP_STACK,
          PUP_WAIT_NC, PUP_WAIT}
```

*9.1.9 M/POST for EBDS **State** Enumeration:*

```
State = {DISCONNECTED, CONNECTING, PUP_ESCROW, IDLING,
         ACCEPTING, ESCROW, STACKING, STACKED, RETURNING,
         RETURNED, REJECTED, JAMMED, STALLED, FAILED,
         CALIBRATE_START, CALIBRATING, DOWNLOAD_START,
         DOWNLOAD_RESTART, DOWNLOADING}
```

*9.1.10 M/POST for EBDS **BNFStatus** Enumeration:*

```
BNFStatus = {UNKNOWN, OK, NOT_ATTACHED, ERROR}
```

*9.1.11 M/POST for EBDS **Bezel** Enumeration:*

```
Bezel = {Standard, Platform, Diagnostic}
```

*9.1.12 M/POST for EBDS **Acceptor** Methods:*

Name/Types	When usable?	Vers
void Open(STRING port_name, PowerUp power_up)	DeviceState == Disconnected.	1.00
void Close(void)	Connected	1.00
void Calibrate(void)	not DeviceBusy and CapCalibrate	1.00
void EscrowReturn(void)	DeviceState == ESCROW or DeviceState == POWER_UP_ESCROW	1.00
void EscrowStack(void)	DeviceState == ESCROW or DeviceState == POWER_UP_ESCROW	1.00
void FlashDownload(STRING file_path)	not DeviceBusy and CapFlashDownload	1.00
void ClearCashBoxTotal(void)	not DeviceBusy and CapCashBoxTotal.	1.00
void SoftReset(void)	CapDeviceSoftReset	1.00
void SetAssetNumber(STRING asset)	not DeviceBusy and CapAssetNumber.	1.00
void SetBezel(Bezel b)	not DeviceBusy and CapSetBezel.	1.10
void SpecifyEscrowTimeout(INT32 bill_timeout, INT32 barcode_timeout)	CapEscrowTimeout	1.00
void SpecifyPupExt(CHAR pup_mode, tPupExt pre_escrow, tPupExt at_escrow, tPupExt post_escrow tPupExt pre_stack)	not DeviceBusy and CapPupExt.	1.00

*9.1.13 M/POST for EBDS **Acceptor** Events:*

Name	Parameter Data	Vers
CONNECTED	none	1.00
DISCONNECTED	none	1.00
ESCROW	none	1.00
PUP_ESCROW	none	1.00
STACKED	none	1.00
RETURNED	none	1.00
REJECTED	none	1.00
CHEATED	none	1.00
CALIBRATE_START	none	1.00
CALIBRATE_PROGRESS	none	1.00
CALIBRATE_FINISH	none	1.00
DOWNLOAD_START	INT32 total_num	1.00
DOWNLOAD_RESTART	none	1.00
DOWNLOAD_PROGRESS	INT32 sector_num	1.00
DOWNLOAD_FINISH	BOOLEAN success	1.00

Name	Parameter Data	Vers
PAUSE_DETECTED	none	1.00
PAUSE_CLEARED	none	1.00
STALL_DETECTED	none	1.00
STALL_CLEARED	none	1.00
JAM_DETECTED	none	1.00
JAM_CLEARED	none	1.00
CASHBOX_REMOVED	none	1.00
CASHBOX_INSTALLED	none	1.00
POWER_UP	none	1.00

*9.1.14 M/POST for EBDS **Acceptor** Properties Details:*

Name		Type	Vers	Access	When usable?
ApplicationID		STRING	1.00	R	CapApplicationID
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property string contains the application ID. This takes the form of a string. For more details see section 7.4.15. If CapApplicationID is false, the value is the empty string.

Name		Type	Vers	Access	When usable?
ApplicationPN		STRING	1.00	R	CapApplicationPN
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property string contains the application part number. This takes the form of a string. For more details see section 7.4.8. If CapApplicationPN is false, the value is the empty string.

Name		Type	Vers	Access	When usable?
AuditLifeTimeTotals		INT32[ ]	1.00	R	CapAudit.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property contains an array of integers with lifetime data. For more details, see section 7.4.11. If this property is unavailable, the value is an empty array.

Name		Type	Vers	Access	When usable?
AuditPerformance		INT32[ ]	1.00	R	CapAudit.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property contains an array of integers with performance data. For more details, see section 7.4.13. If this property is unavailable, the value is an empty array.

Name		Type	Vers	Access	When usable?
AuditQP		INT32[ ]	1.00	R	CapAudit.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property contains an array of integers with performance data. For more details, see section 7.4.12. If this property is unavailable, the value is an empty array.

Name		Type	Vers	Access	When usable?
AutoStack		BOOLEAN	1.00	R/W	Connected.
Read point:	Local host variable.		**Write point:**		Immediately when written.

This property controls the handling of documents. If set, documents are automatically stacked when they are received. If not set, the application is informed via the ESCROW event when a document arrives. For more details see section 8.2.3. If this property is unusable, it is ignored.

Name		Type	Vers	Access	When usable?
BarCode		STRING	1.00	R	DocType == DocBarCode.
Read point:	At barcode escrow.		**Write point:**		n/a

This property contains the barcode info extracted from the most recent bar-coded document. See section 7.1.3 for more details. If a bar-coded document is not being processed, this property has an undefined value.

Name		Type	Vers	Access	When usable?
Bill		Bill	1.00	R	DocType == DocBill.
Read point:	At bill escrow.		**Write point:**		n/a

This property contains the bill info extracted from the most recent bank note. See sections 7.1.4, 8.2 and 9.1.2 for more details. If a bank note is not being processed, this property has an undefined value.

Name		Type	Vers	Access	When usable?
BillTypes		Bill[ ]	1.00	R	Connected.
Read point:	Cached on connect.		**Write point:**		n/a

This property is an array of all of the bill's that are accepted by the device. This includes entries for each variety of bank note. This table is constructed when the connection to the acceptor is established. See section 9.1.2 for more details. If this property is unavailable, the value is an empty array.

An example of a print out of this property for a bill acceptor loaded with the US dollar variant part number 490320223 is shown below:

```
1 USD        1 C A B B
2 USD        2 C A B A
3 USD        2 C B B A
```

```
 4 USD       5 C A B B
 5 USD       5 D A B C
 6 USD      10 D A B B
 7 USD      10 E A B C
 8 USD      10 F A B B
 9 USD      20 C A B B
10 USD      20 D A B C
11 USD      20 E A B B
12 USD      50 C A B B
13 USD      50 D A B C
14 USD      50 D B B C
15 USD      50 F A B B
16 USD     100 C A B B
17 USD     100 D A B D
18 USD     100 D B B C
```

Name		Type	Vers	Access	When usable?
BillTypeEnables		BOOLEAN[ ]	1.00	R/W	Connected.
Read point:	Created on connect. Pre-filled with true entries.		**Write point:**		Sent to the bill acceptor at the next available time slot.

This property is an array of Boolean values that correspond to the entries in the BillTypes property. Acceptance of bills that correspond to those values may be controlled with true entries accepting notes and false entries rejecting them. See sections 7.1.1 and 7.5.2 for more details. If this property is unavailable, the value is an empty array.

Note: Changes in the BillValueEnables property will result in changes to the BillTypesEnables property. However, changes made to the BillTypesEnables do NOT propagate back to the BillValueEnables property. In general, the application should use one of the bill enable properties and not switch back and forth between them as this may cause unexpected results.

ERRATA: On some bill acceptors with out-of-date firmware there is a defect in the code that will cause the units to not be enabled until the BillTypeEnables property is set. To get around this defect the application may place a line of code that sets this property in its Connected event handler. An example, harmless line of code might look like:

```
billAcceptor.BillTypeEnables = billAcceptor.BillTypeEnables;
```

Name		Type	Vers	Access	When usable?
BillValues		Bill[ ]	1.00	R	Connected.
Read point:	Cached on connect.		**Write point:**		n/a

This property is an array of all of the bill's that are accepted by the device. Variations in like-valued bank notes are ignored, so that each entry has a different monetary value or country. This table is constructed when the connection to the acceptor is established. See section 9.1.2 for more details. If this property is unavailable, the value is an empty array.

An example of a print out of this property for a bill acceptor loaded with the US dollar variant part number 490320223 is shown below:

```
1 USD       1 * * * *
2 USD       2 * * * *
3 USD       5 * * * *
4 USD      10 * * * *
5 USD      20 * * * *
6 USD      50 * * * *
7 USD     100 * * * *
```

Note that the view of the bill set is simpler than that in the BillTypes property, but that the detail information about the various bill types and variations is not available.

Name		Type	Vers	Access	When usable?
BillValueEnables		BOOLEAN[ ]	1.00	R/W	Connected.
Read point:	Created on connect. Pre-filled with true entries.		**Write point:**		Sent to the bill acceptor at the next available time slot.

This property is an array of Boolean values that correspond to the entries in the BillValues property. Acceptance of bills that correspond to those values may be controlled with true entries accepting notes and false entries rejecting them. See sections 7.1.1 and 7.5.2 for more details. If this property is unavailable, the value is an empty array.

Note: Changes in the BillValueEnables property will result in changes to the BillTypesEnables property. However, changes made to the BillTypesEnables do NOT propagate back to the BillValueEnables property. In general, the application should use one of the bill enable properties and not switch back and forth between them as this may cause unexpected results.

Name		Type	Vers	Access	When usable?
BNFStatus		BNFStatus	1.00	R	CapBNFStatus
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property can be used to determine the status of the optional bunch note feeder attachment. This property is only valid when CapBNFStatus is true. When false, the status will always be UNKNOWN.

Name		Type	Vers	Access	When usable?
BootPN		STRING	1.00	R	CapBootPN.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property string contains the boot part number. This takes the form of a string. For more details see section 7.4.7. If CapBootPN is false, the value is the empty string.

Name		Type	Vers	Access	When usable?
Cap***		BOOLEAN	1.00	R	Connected.
Read point:	Cached on connect.		**Write point:**		n/a

The properties beginning with “Cap” are all capability flags that indicate what features are present on the connected bill acceptor. For the most part, these flags are derived from the model number field sent by the acceptor (see section 7.1.2)

Name	Model #	Other Criteria
CapApplicationID		Device Capability Map is set for query acceptor application ID.
CapApplicationPN	“T”, “U”	
CapAssetNumber	“T”, “U”	
CapAudit	“T”, “U”	
CapBarCodes	15, 23 , “T”, “U”	
CapBarCodesExt		The RawBarcodes bit is set in device replies (see section 7.1.2)
CapBNFStatus		Device Capability Map is set for query BNF status.
CapBookmark		True
CapBootPN	“T”, “U”	
CapCalibrate		True
CapCashBoxTotal	“A”, “B”, “C”, “D”, “G”, “M”, “P”, “W”, “X”	
CapCouponExt	“P”, “X”	
CapDevicePaused	31, “P”, “X”	
CapDeviceSoftReset	31, “A”, “B”, “C”, “D”, “G”, “M”, “P”, “T”, “U”, “W”, “X”	
CapDeviceType	“T”, “U”	
CapDeviceResets	“A”, “B”, “C”, “D”, “G”, “M”, “P”, “T”, “U”, “W”, “X”	
CapDeviceSerialNumber	“T”, “U”	
CapEscrowTimeout	“T”, “U”	
CapFlashDownload		True
CapNoPush	23, 31, “P”, “X”	
CapOrientationExt		Device Capability Map is set for extended orientation control.
CapPupExt		Device Capability Map is set for extended PUP mode.
CapSetBezel		Device Capability Map is set for set bezel.
CapTestDoc		Device Capability Map is set for test document accepted.
CapVariantID		Device Capability Map is set for query acceptor variant ID.
CapVariantPN	“T”, “U”	

See section 7.4.11 for a description of the Device Capability Map.

Name		Type	Vers	Access	When usable?
CashBoxAttached		BOOLEAN	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

This property is true when a cash box is attached to the bill acceptor. See section 7.1.2 for more details on this flag.

Name		Type	Vers	Access	When usable?
CashBoxFull		BOOLEAN	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

This property is true when a cash box is full and can no longer accept bills. See sections 7.1.2 and 8.4.6 for more details on this flag.

Name		Type	Vers	Access	When usable?
CashBoxTotal		INT32	1.00	R	not DeviceBusy and CapCashBoxTotal.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property contains reflects the amount of currency believed to present in the cash box. See section 7.4.2 for more details.

Name		Type	Vers	Access	When usable?
Connected		BOOLEAN	1.00	R	Always.
Read point:	Updated on each poll.		**Write point:**		n/a

The connected property is true if a device is connected and responding to the host system. This becomes false when the device does not respond to polls for more than an allowable limit, or the connection is closed by the host. This limit is five seconds for model “T” and “U” devices and thirty seconds for all others.

Name		Type	Vers	Access	When usable?
Coupon		Coupon	1.00	R	DocType == DOC_COUPON
Read point:	At coupon escrow.		**Write point:**		n/a

This property contains the coupon info extracted from the most recent generic coupon. See sections 7.1.5, and 9.1.3 for more details. If a coupon note is not being processed or EnableCouponExt is false, this property has an undefined value.

Name		Type	Vers	Access	When usable?
DebugLog		BOOLEAN	1.00	R/W	Always.
Read point:	Local host variable.		**Write point:**		Set by application code.

This property is used to control the generation of a debug log file. In the development phase of an application, this log can be useful in diagnosing any problems or issues that might arise.

Note that the debug log file is locked while the application is using it. This flag should be false for deployed applications.

Name		Type	Vers	Access	When usable?
DebugLogPath		STRING	1.00	R/W	Read: Always Write: When DebugLog is false.
Read point:	Local host variable.		**Write point:**		Set by application code.

This property is used to control the location of the debug log file. When the DebugLog property is set to true, the value of this property is used to determine the location of the log file. By default, the log file is created in the same folder where the application is deployed.

Name		Type	Vers	Access	When usable?
DeviceBusy		BOOLEAN	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

This flag is true if the bill acceptor is in the idle state. In this state, it is possible to perform more complex commands that would interfere with bill acceptance.

Name		Type	Vers	Access	When usable?
DeviceCRC		INT32	1.00	R	Connected.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This value is the CRC of the flash memory (excluding boot and “special” areas). See section 7.4.1 for more details.

Name		Type	Vers	Access	When usable?
DeviceFailure		BOOLEAN	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

When this flag is set, the bill acceptor is out of service and requires attention, typically in the form of a call by a service technician. See sections 7.1.2 and 8.4.12 for more details on this flag.

Name		Type	Vers	Access	When usable?
DeviceJammed		BOOLEAN	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

When this flag is set, the bill acceptor has encountered a jam condition and requires attention, typically in the form of a clearing debris from the bill path. See section 7.1.2 for more details on this flag.

Name		Type	Vers	Access	When usable?
DeviceModel		INT32	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

This number is set from the model number field returned by the bill acceptor for each poll. If the value is between 32 and 126, it should be treated as a single printable character. See section 7.1.2 for more details on this value.

Name		Type	Vers	Access	When usable?
DevicePaused		BOOLEAN	1.00	R	CapDevicePaused.
Read point:	Updated on each poll.		**Write point:**		n/a

This flag is set when the bill acceptor is in the PAUSED condition. This occurs when the consumer attempts to insert another bill while the previous one is still being processed. The system pauses to avoid grabbing two bills and "stealing" the second one. See section 7.1.2 for more details on this value. If this property is not supported, it will always be false.

Name		Type	Vers	Access	When usable?
DevicePortName		STRING	1.00	R	Connected.
Read point:	Cached on connect.		**Write point:**		n/a

This is a copy of the port_name parameter passed into the Open method. It is the serial or virtual serial port used to communicate with the bill acceptor.

Name		Type	Vers	Access	When usable?
DevicePowerUp		PowerUp	1.00	R	Connected.
Read point:	Cached on connect.		**Write point:**		n/a

This is a copy of the power_up parameter passed into the Open method. It is of type tPowerUp and control the behavior of the bill acceptor when a bill is being processed during a power failure. See section 7.1.1 for more details.

Name		Type	Vers	Access	When usable?
DeviceResets		INT32	1.00	R	CapDeviceResets.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property returns the number of times that the bill acceptor has been reset. See section 7.4.3 for more details. If this property is not supported, it will always be 0.

Name		Type	Vers	Access	When usable?
DeviceRevision		INT32	1.00	R	Connected.
Read point:	Updated on each poll.		**Write point:**		n/a

This property is the numeric value contained in the standard EBDS revision field. See section 7.1.2 for more details.

Name		Type	Vers	Access	When usable?
DeviceSerialNumber		STRING	1.00	R	CapDeviceSerialNumber.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property is the serial number of the attached device in string format. See section 7.4.6 for more details on this property. If this property is not supported, it will always be the empty string.

Name		Type	Vers	Access	When usable?
DeviceSignature		INT32	1.00	R	Connected.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property returns a value that corresponds to the legacy mode device signature outlined in section 8.3.1. This signature can be used to affirm that the code contained within a unit matches the expected code.

Name		Type	Vers	Access	When usable?
DeviceStalled		BOOLEAN	1.00	R	CapNoPush.
Read point:	Updated on each poll.		**Write point:**		n/a

This flag is set when the bill acceptor is in the STALLED condition. This occurs when a jam is detected and NoPush mode is active. The system pauses so that the operator can examine the bill that caused the jam and to check for a possible fraud. See sections 7.1.2 and 8.4.13 for more details on this value. If this property is not supported, it will always be false.

Name		Type	Vers	Access	When usable?
DeviceState		State	1.00	R	Always.
Read point:	Local host variable.		**Write point:**		n/a

This variable reflects the current state of the bill acceptor. See section 9.1.9 as well a section 7.1.2 for details.

Name		Type	Vers	Access	When usable?
DeviceType		STRING	1.00	R	CapDeviceType
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property is the device type of the attached device in string format. See section 7.4.5 for more details on this property. If this property is not supported, it will always be the empty string.

Name		Type	Vers	Access	When usable?
DocType		DocType	1.00	R	After DeviceState == Escrow.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property is the type of document currently being processed. Typically this property is interrogated at escrow and when a document is stacked to determine how it should be processed. See section 9.1.4 for a list of possible document types. If this property is not current, it will reflect the value of the last document processed by the acceptor.

Name		Type	Vers	Access	When usable?
EnableAcceptance		BOOLEAN	1.00	R/W	Connected.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control acceptance of documents by the bill acceptor. If false, the bill acceptor will accept no documents, if true, then the selected bills, and other documents will be accepted.

Name		Type	Vers	Access	When usable?
EnableBarCodes		BOOLEAN	1.00	R/W	CapBarCodes.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control acceptance of bar coded documents. If false, the bill acceptor will accept no bar codes, if true, then bar codes will be accepted if the EnableAcceptance property is true. This property is ignored if CapBarCodes is false.

Name		Type	Vers	Access	When usable?
EnableBookmarks		BOOLEAN	1.00	R/W	CapBookmark.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control acceptance of book mark documents. If false, the bill acceptor will accept no book marks, if true, then book marks will be accepted if the EnableAcceptance property is true. This property is ignored if CapBookmark is false.

Name		Type	Vers	Access	When usable?
EnableCouponExt		BOOLEAN	1.00	R/W	CapCouponExt
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control acceptance of generic coupon documents. If false, the bill acceptor will treat the coupons the same as a bill of the same value. If true, then coupons receive special processing and more details about the coupon are available vie the Coupon property. This property is ignored if CapCouponExt is false.

Name		Type	Vers	Access	When usable?
EnableNoPush		BOOLEAN	1.00	R/W	CapNoPush.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control the handling of jam conditions that require a stack of the offending document. If false, the bill acceptor will function normally, if true, the acceptor will enter the STALLED condition when a jam recovery needs to stack. This property is ignored if CapBookmark is false.

Name		Type	Vers	Access	When usable?
EscrowOrientation		Orientation	1.00	R	After DeviceState == Escrow. when CapOrientationExt.
Read point:	At bill escrow.		**Write point:**		n/a

This property reflects the orientation of bank notes that are fed into the bill acceptor. Note that bar codes, coupons or bookmarks do not have orientation data. The orientation values are specified in section 9.1.5. Note that if CapOrientationExt is false then the orientation returned is always ORIENTATION_UNKNOWN.

Name		Type	Vers	Access	When usable?
HighSecurity		BOOLEAN	1.00	R/W	Connected.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control the security criteria applied to the processing of bills. This is a somewhat obsolete concept in that bill acceptance is normally optimize for maximum acceptance and security. Thus most bill acceptors ignore this value. See section 7.1.1 for more details on this setting.

Name		Type	Vers	Access	When usable?
OrientationCtl		OrientationCtl	1.00	R/W	Connected.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control the orientation criteria applied to the processing of bills. See sections 7.1.1 and 8.2.4 for more details on this setting.

Name		Type	Vers	Access	When usable?
OrientationCtlExt		OrientationCtl	1.00	R/W	CapOrientationExt.
Read point:	Local host variable.		**Write point:**		Updated on each poll.

This property is used to control the orientation criteria applied to the processing of bills. See sections 7.1.1, 8.2.4 and 8.2.5 for more details on this setting. If CapOrientationExt is false, this property is ignored.

Name		Type	Vers	Access	When usable?
VariantID		STRING	1.00	R	CapVariantID.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property string contains the bill variant ID. This takes the form of a string. For more details see section 7.4.10. If CapVariantID is false, the value is the empty string.

Name		Type	Vers	Access	When usable?
VariantNames		STRING[ ]	1.00	R	Connected.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property is an array of strings that represent the country codes of the currencies accepted by the bill acceptor. See section 7.4.9 for more details.

Name		Type	Vers	Access	When usable?
VariantPN		STRING	1.00	R	CapVariantPN.
Read point:	Query acceptor on demand.		**Write point:**		n/a

This property string contains the bill variant part number. This takes the form of a string. For more details see section 7.4.10. If CapVariantPN is false, the value is the empty string.

Name		Type	Vers	Access	When usable?
Version		STRING	1.00	R	Always
Read point:	Constant data.		**Write point:**		n/a

This property contains the version string on the M/POST code. For version 1.00 this is the string “V1.00, 283792100”.

*9.1.15 M/POST for EBDS **Acceptor** Methods Details:*

```
void Open(STRING port_name, tPowerUp power_up)
```

This function is used to open a connection to the bill acceptor. This function may only be called when the bill acceptor is in the DISCONNECTED state. On return, the process of connecting will be started. This process is completed when the CONNECTED event is sent to the application and the state of the bill acceptor moves passed the CONNECTING state.

If a connection cannot be established, the DISCONNECTED event is sent to the application to indicate failure.

If the unit is “stuck” in download mode, the DOWNLOAD_RESTART event is sent so the application can restart the flash download or signal an error if that option is not supported.

Exception(s):
INVALID_PORT, INVALID_STATE

```
void Close(void)
```

This function is used to close the connection to the bill acceptor. This function encapsulates all of the application session closing requirements outlined in section 6.3. When this process is completed, the DISCONNECTED event is sent to the application.

Exception(s):
INVALID_STATE

```
void Calibrate(void)
```

This function is used to start the calibrate process. When the bill acceptor is ready to accept the calibration document, the CALIBRATE_START will be sent to the application. As calibration progresses, the CALIBRATE_PROGRESS event informs the host. When the calibration is completed, the CALIBRATE_FINISH event informs the application.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND

```
void EscrowReturn(void)
```

This function is used to return the document currently at escrow. This can only be done when the state is ESCROW or PUP_ESCROW.

Exceptions(s)
INVALID_STATE

```
void EscrowStack(void)
```

This function is used to stack a document currently at escrow. This can only be done when the state is ESCROW or PUP_ESCROW.

Exceptions(s)
INVALID_STATE

```
void FlashDownload(STRING file_path)
```

This function is used to start the process of updating the flash memory of the bill acceptor. Progress in the download is signaled with the events DOWNLOAD_START, DOWN_LOAD_PROGRESS and DOWNLOAD_FINISH.

Exception(s):
INVALID_STATE, FILE_NOT_FOUND, INVALID_FILE

```
void ClearCashBoxTotal(void)
```

This function is used to clear the count of bills stored in the cash box. This function may only be called if the not DeviceBusy and CapCashBoxTotal.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND, INVALID_ARG

```
void SetAssetNumber(STRING asset)
```

This function is used to set the asset number of the bill acceptor and the cash box. See section 7.5.4 for more details.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND, INVALID_ARG

```
void SpecifyEscrowTimeout(INT32 bill_timeout,
                          INT32 barcode_timeout)
```

This function is used to specify the escrow timeout parameters in use by the bill acceptor. There are two settings, one for bank notes and the other for bar codes. Both are in seconds from 0 through 127 where 0 represents no timeout. See section 7.5.6 for more details.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND, INVALID_ARG

```
void SpecifyPupExt(CHAR pup_mode,
                   PupExt pre_escrow,
                   PupExt at_escrow,
                   PupExt post_escrow,
                   PupExt pre_stack)
```

This function is used to specify the parameters in use by the bill acceptor to recover from when a bill is being processed during a power fail. See section 7.5.3 for more details.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND

```
void SetBezel(Bezel b)
```

This function is used to override the default bezel setting. Note that setting the bezel to the wrong type may result in unpredictable bezel ooperation.

Exception(s):
INVALID_STATE, UNSUPPORTED_COMMAND

*9.1.16 M/POST for EBDS **Acceptor** Events Details:*

Name	Parameter Data	Vers
CONNECTED	none	1.00

This event is sent when the connection to the bill acceptor has been established.

Name	Parameter Data	Vers
DISCONNECTED	none	1.00

This event is sent when the connection to the bill acceptor has been terminated or lost. For the latter, see section 8.4.2 for more details on dealing with this event.

Name	Parameter Data	Vers
ESCROW	none	1.00

This event is sent when a document reaches escrow. The doc_type parameter identifies the type of document. It is important that all types of documents be processed, even if just to be returned to the consumer. Stranding a document in Escrow can cause the bill acceptor to be unable to accept further notes.

Name	Parameter Data	Vers
PUP_ESCROW	DocType doc_type	1.00

This event is sent when a document is “found” at escrow during power up. The doc_type parameter identifies the type of document. It is important that all types of documents be processed, even if just to be returned to the consumer. Stranding a document in Escrow can cause the bill acceptor to be unable to accept further notes.

Name	Parameter Data	Vers
STACKED	DocType doc_type	1.00

This message is sent when a document is stacked. It is at this time that the application should fix credit to the transaction. Any delivery of products and services should not proceed until this event is signaled with adequate credit.

Name	Parameter Data	Vers
RETURNED	none	1.00

This message is sent when a document is returned to the customer.

Name	Parameter Data	Vers
REJECTED	none	1.00

This message is sent when a document is rejected by the bill acceptor. See section 8.4.4 for more details on dealing with this event.

Name	Parameter Data	Vers
CHEATED	none	1.00

This message is sent when difficulty is encountered transporting a document that may have been caused by a cheat attempt. See section 8.4.3 for more details on dealing with this event. MEI does not offer a method to set / test cheat events.

Name	Parameter Data	Vers
CALIBRATE_START	none	1.00

This message is sent when the calibration process is started and the user needs to insert the calibration document.

Name	Parameter Data	Vers
CALIBRATE_PROGRESS	none	1.00

This event signals that calibration is on going. See section 8.4.9 for more details on dealing with this event.

Name	Parameter Data	Vers
CALIBRATE_FINISH	none	1.00

This event signals that calibration has completed.

Name	Parameter Data	Vers
DOWNLOAD_START	INT32 total_num	1.00

This event signals that the download of code has started and that total_num sectors need to be sent to the bill acceptor.

Name	Parameter Data	Vers
DOWNLOAD_RESTART	none	1.00

This event indicates that the bill acceptor power up in download mode. The host system needs to restart the download process.

Name	Parameter Data	Vers
DOWNLOAD_PROGRESS	INT32 sector_num	1.00

This event signals that the flash download is on going and that sector_num sectors have been sent.

Name	Parameter Data	Vers
DOWNLOAD_FINISH	BOOLEAN success	1.00

This event indicates that the download has ended. The success parameter is true for success and false for failure.

Name	Parameter Data	Vers
PAUSE_DETECTED	none	1.00

This is sent when the bill acceptor is paused because the customer is feeding bills too quickly. See section 8.4.8 for more details on dealing with this event.

Name	Parameter Data	Vers
PAUSE_CLEARED	none	1.00

This is sent when the bill acceptor bill path is cleared by the customer. See section 8.4.8 for more details on dealing with this event.

Name	Parameter Data	Vers
STALL_DETECTED	none	1.00

This event is sent when the bill acceptor becomes stalled after an anti-jam effort. The bill acceptor requires attention to resolve the jam. See section 8.4.13 for more details on dealing with this event.

Name	Parameter Data	Vers
STALL_CLEARED	none	1.00

This event is sent when the bill acceptor stall is cleared after operator intervention. The error condition is cleared and the application can proceed.

Name	Parameter Data	Vers
JAM_DETECTED	none	1.00

This event is sent when the bill acceptor detects a jam. The bill acceptor requires attention to resolve the jam. See section 8.4.5 for more details on dealing with this event.

Name	Parameter Data	Vers
JAM_CLEARED	none	1.00

This event is sent when the bill acceptor jam is cleared, either automatically or after operator intervention. The error condition is cleared and the application can proceed.

Name	Parameter Data	Vers
CASHBOX_REMOVED	none	1.00

This event is sent when the cash box is removed from the bill acceptor. This should only happen as part of regular procedures when the money is being collected. See section 8.4.7 for more details on dealing with this event.

Name	Parameter Data	Vers
CASHBOX_INSTALLED	none	1.00

This event is sent when the cash box is returned to the bill acceptor. This should only happen as part of regular procedures when the money is being collected. See section 8.4.7 for more details on dealing with this event.

Name	Parameter Data	Vers
POWER_UP	none	1.00

This event is sent when the bill acceptor has powered up or been reset. See section 8.4.10 for more details on dealing with this event.

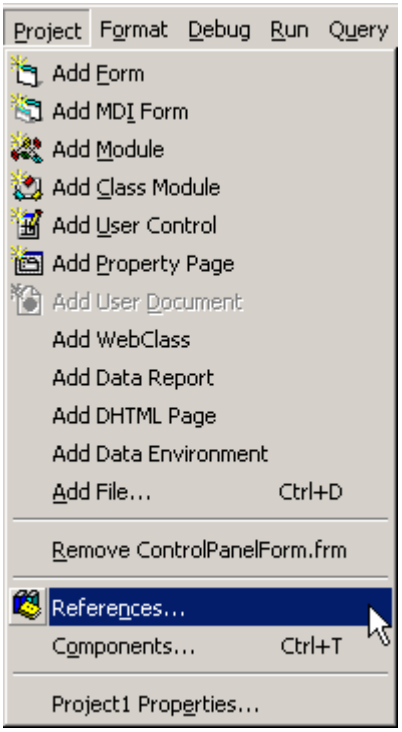
10. M/POST Bindings for ActiveX

The goal of this section is to provide OLE specific information for developers in that environment, especially developers using the Visual Basic 6 programming language.

10.1 Connecting to the M/POST DLL

Before you can use the M/POST OLE component from Visual Basic or another development environment, the MPOST_OLE.dll must be registered. If you install M/POST OLE using the installer program, the installer will register the DLL. If you do you use to installer but instead simply copy MPOST_OLE.dll to your chosen directory, you will need to execute the command line “regsvr32 MPOST_OLE.dll”. Depending upon the configuration of your sytem, you might need to specify the full path for regsvr32.exe. This utility should be located in one of your Windows directories.

Once MPOST_OLE.DLL is registered, you will need to add a reference to the component to your Visual Basic project. To do this, select the References command from the Project menu and select MPOST OLE 1.0 Type Library (the version number might be greater than 1.0).



After adding the reference, you will be able to declare and use an instance of the Acceptor object or any of the other M/POST objects, enumerations or declarations.

10.2 Handling Events in Visual Basic 6

Visual Basic 6 makes it relatively simple to handle events sent by M/POST OLE, but you must remember to declare the Acceptor object using the WithEvents keyword.

```
Public WithEvents Acceptor As MPOST_OLELib.Acceptor
```

When you do this, Visual Basic will add “Acceptor” to the object list (the drop-down list on the left hand side above the text pane). If you select “Acceptor” from this list, you will see the list of all supported events in the drop-down list on the right hand side.

Simply select the desired event from the list, and Visual Basic will automatically add a handler subroutine for you.

```
Private Sub Acceptor_Connected()
      ‘Add your handler code here.
End Sub
```

10.3 Differences from the M/POST model

Due to the requirements of the OLE programming environment, some changes to the M/POST model presented in section 9 were required. These changes are summarized below:

- To conform to convention, the OrientationCtrl property is called OrientationControl to match the OrientationControl enumeration.

- For the BNFStatus Enumeration:

Model Value	**OLE Binding**
Unknown	BNFUnknown
Ok	BNFOK
NotAttached	BNFNotAttached
Error	BNFError

- For the DocType Enumeration:

Model Value	**OLE Binding**
None	DocNone
NoValue	DocNoValue
Bill	DocBill
Barcode	DocBarcode
Coupon	DocCoupon

- The Orientation Enumeration

Model Value	**OLE Binding**
RightUp	RightUp
RightDown	RightDown
LeftUp	LeftUp
LeftDown	LeftDown
Unknown	OrientationUnknown

- The PowerUp Enumeration

Model Value	OLE Binding
A	PUP_A
B	PUP_B
C	PUP_C
E	PUP_E

10.4 A note on array index values.

For consistency with the M/POST model, arrays are zero-based, rather than one-based as is the usual Visual Basic convention. Thus valid indexes run from 0 through N-1, rather than 1 through N for an array of N elements.

10.5 A note on Boolean property values.

For constancy with the M/POST model, Boolean property values use 0 for false and 1 for true, This is different from VB6 custom. For simple testing of a property there is no impact, however if more complex operations are required it will be necessary to compare the value with 1 as in the following example:

```
If cint(Acceptor.Connected) = 0 Then ...
```

instead of

```
If NOT Acceptor.Connected Then ...
```

For greatest compatibility, only compare with 0. This way, if the M/POST library value for TRUE should be changed, the application code will continue to run correctly. For example use

```
If cint(Acceptor.Connected) <> 0 Then ...
```

instead of

```
If cint(Acceptor.Connected) = 1 Then ...
```

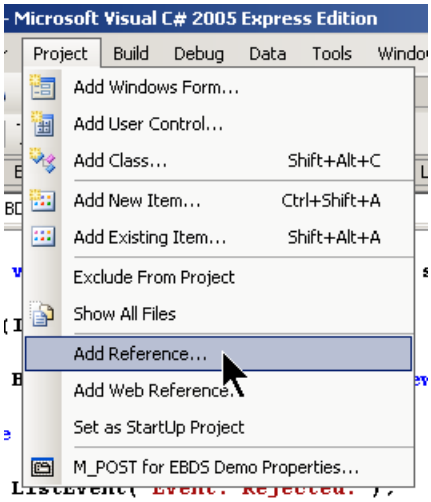
because the latter will break if the value of TRUE is changed. The value of FALSE will always be 0.

11. M/POST Bindings for .NET

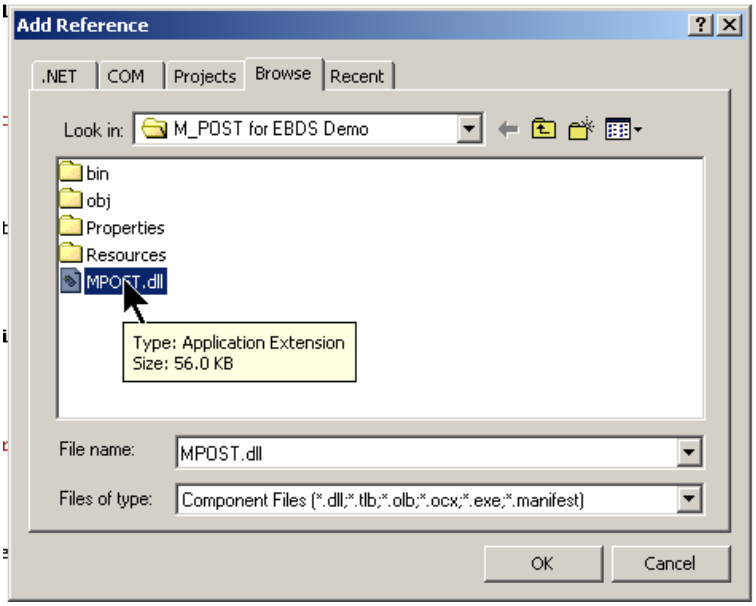
The goal of this section is provided to supply .NET specific information for developers in that environment.

11.1 Connecting to the M/POST DLL

The full release version of M/POST shall have a full install system. The Early Experience version does not. In order to utilize M/POST is an application, it is necessary to copy the MPOST.DLL file into the project folder and "Add a Reference" to it. Under the Project menu entry:



And then browse for the DLL.



11.2 Handling M/POST Events in C#

When M/POST sends events to a C# application, they are handled with delegates. It is recommended that delegates are declared as private data as shown below:

```
private EscrowEventHandler EscrowedDelegate;
```

and that it be connected to the handler as follows:

```
// Form Window Constructor Function.
public EBDS_Bill_Acceptor()
{
        InitializeComponent();

        // Initialize the deligates.
        EscrowedDelegate = new EscrowEventHandler(HandleEscrowedEvent);
        // Balance removed for clarity.

        BillAcceptor.OnEscrow += EscrowedDelegate;
        // Balance removed for clarity.
}
```

In the handler itself, the application programmer needs to deal with the fact that M/POST events are sent from a worker thread. This means that:

a) The user interface is not directly accessible due it's single threaded nature.
b) That if a command is sent, or a property is accessed, directly from the same thread, the system could lock up in a "deadly embrace".

To resolve both of these problems, always use the .NET Invoke mechanism to process events. This is shown below:

```
private void HandleEscrowedEvent(object sender, EventArgs e)
{
        if (InvokeRequired)
        {
                BeginInvoke(EscrowedDelegate, new object[] { sender, e });
        }
        else
        {
                // Event handling code goes here!
        }
}
```

If the receiver of events is not a kind of form window, it will need access to a form window or other window control for this to work. For example:

```
private void HandleEscrowedEvent(object sender, EventArgs e)
{
        if (MainWin.InvokeRequired)
        {
                MainWin.BeginInvoke(EscrowedDelegate, new object[] { sender, e });
        }
        else
        {
                // Event handling code goes here!
        }
}
```

11.3 Handling M/POST Events in VB.NET

When M/POST sends events to a VB.NET application, they are handled with delegates. It is recommended that delegates are declared as private data as shown below:

```
Private EscrowedDelegate As EscrowEventHandler
```

and that it be connected to the handler as follows:

```
' Form Window Constructor Function.
Public Sub New()
        InitializeComponent()

        ' Initialize the deligates.
        EscrowedDelegate = New EscrowEventHandler(AddressOf HandleEscrowedEvent)
        ' Balance removed for clarity.

        ' Connect to the events.
        AddHandler BillAcceptor.OnEscrow, EscrowedDelegate
        ' Balance removed for clarity.
End Sub
```

In the handler itself, the application programmer needs to deal with the fact that M/POST events are sent from a worker thread. This means that:

c) The user interface is not directly accessible due it's single threaded nature.
d) That if a command is sent, or a property is accessed, directly from the same thread, the system could lock up in a "deadly embrace".

To resolve both of these problems, always use the .NET Invoke mechanism to process events. This is shown below:

```
Private Sub HandleEscrowedEvent(ByVal sender As Object, ByVal e As EventArgs)
        If InvokeRequired Then
                BeginInvoke(EscrowedDelegate, New Object() { sender, e })
        Else
                ' Event handling code goes here!
        End If
End Sub
```

12. M/POST Bindings for Linux

The goal of this section is provided to supply Linux specific information for developers in that environment.

12.1. Using the MPOST_Linux Library

To install the M/POST Linux library, simply uncompress the archive MPOST_Linux.tar.gz.

You will need to build the library and, if desired, the demo program. These projects were created with Eclipse. If you have Eclipse, simply import the projects into your workspace and build.

If you do not have Eclipse, you can either create corresponding projects in the IDE you are using, or you can build from the command line. It is possible that the makefiles in the Debug and Release directories will work. If not, you will need to compile the files using gcc or another C++ compiler. In order to build the MPOST_Linux_Demo project, you will need additional project settings, described below.

The C++ project MPOST_Linux is configured to produce a static library named MPOST_Linux (actual filename libMPOST_Linux.a).

All of the M/POST functions you will need to access are located in the class CAcceptor. To access these functions, #include the header file Acceptor.h. For convenience, you should also add the following statement:

```
using namespace MPOST;
```

The M/POST documentation in this manual was originally written for the C# language. In some cases there might be differences in syntax for calling functions. Refer to the header file Acceptor.h for exact naming and syntax of functions and enumerations.

12.2. Handling M/POST Events in Linux

In order to handle a CAcceptor event, first define a function with the following signature:

```
void EventHandler(CAcceptor*, int);
```

Note that the int parameter is only used by the DownloadStart and DownloadProgress events.

Set the event handler with a statement like the following:

```
acceptor->SetEventHandler(ConnectedEvent, ConnectedEvenrHandler);
```

At present, each event may only have one handler.

12.3. MPOST Linux Demo Program

We include in addition to the M/POST Linux library a demo program that you can use to verify that M/POST is working correctly and as sample code showing how the library is used.

The most complete version of the demo program (includes samples of most functions) is a GUI application using GTK. If you do not have GTK available, you can still refer to the demo source for samples of calling CAcceptor functions. Additionally, you can build a limited command-line version. Refer to the file MPOST_Linux_Demo/main.cpp for more information.

If you do build using GTK and use Eclipse, the project files provide contain the correct libraries and directories. If you build from the command line, you will have to tell the compile to look for the following include directories:

13. M/POST Bindings for JAVA

A Java binding for M/POST is not planned at this time. To inquire as to any change in this policy please see the MEI web site at www.meiglobal.com.

14. The M/POST Demo Program

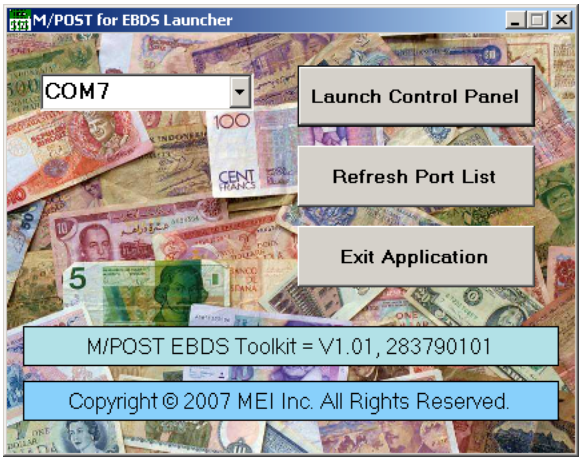
The various versions of M/POST all ship with a demo program that serves multiple purposes:

1. It is a test bed for validating the customer installation of the hardware and software.
2. It is a useful vehicle for familiarizing application developers with the capabilities of the bill acceptor hardware and the library software.
3. The source code provides examples of the handling of events and other coding tasks required of the application developer.
4. The demo program served as a test bed for developing, test and debugging the library

While there will be several different versions of the demo program, all will be based on and similar to the .NET demo program presented here. Where significant differences occur, they will be highlighted in the text.

14.1 The Launcher

The launcher screen is the initial window displayed by the application. It is shown below:



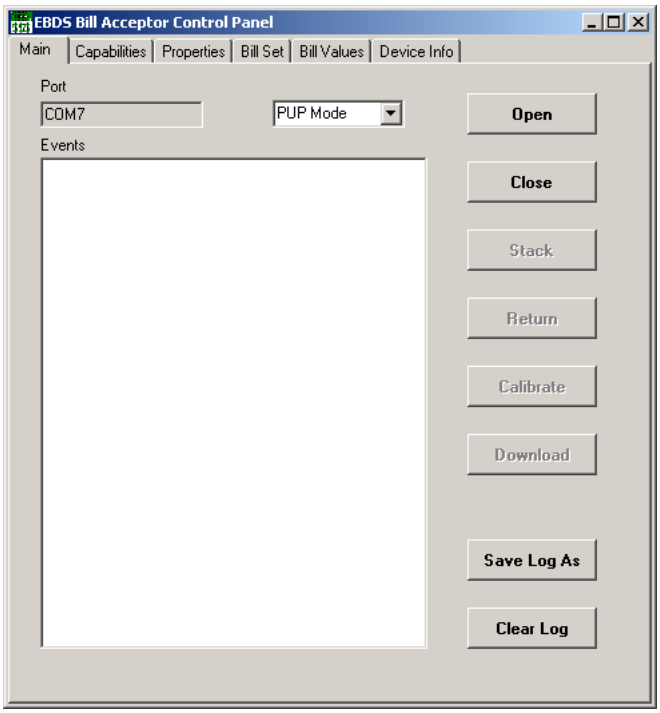
In this simple dialog box, the controls are used as follows:

- The combo-box on the left contains a list of available ports.
- The Launch Control Panel button opens a control panel on the com port selected in the combo-box.
- The Refresh Port List button updates the combo-box for cases when ports are added on the fly (e.g. plugging in a USB port or device).
- The Exit Application button exits the application. There was an effort made to make this seem more dramatic, but it sorta fell through.

Note that in the second text bar from the bottom, the version and part number of the M/POST DLL are displayed. When reporting issues or requesting help, it is important to reference this part number. The bottom text bar is a reminder that this is MEI code and not public domain. Finally the background bitmap is meant to be suggestive of the general topic of the program.

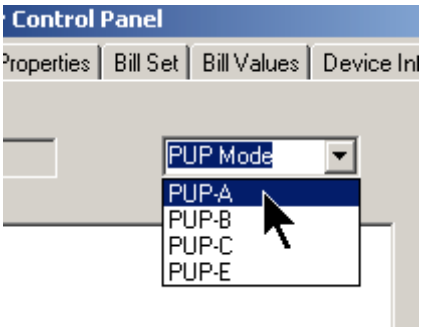
14.2 The Control Panel

The control panel for an EBDS bill acceptor is a multi-tabbed dialog box that permits a great deal of control and provides detailed information on the attached bill acceptor. When the control panel is first displayed, it is in the Main tab, and while the panel is open, the connection to the bill acceptor is not. This is illustrated below.



In this condition there is not much that can be done.

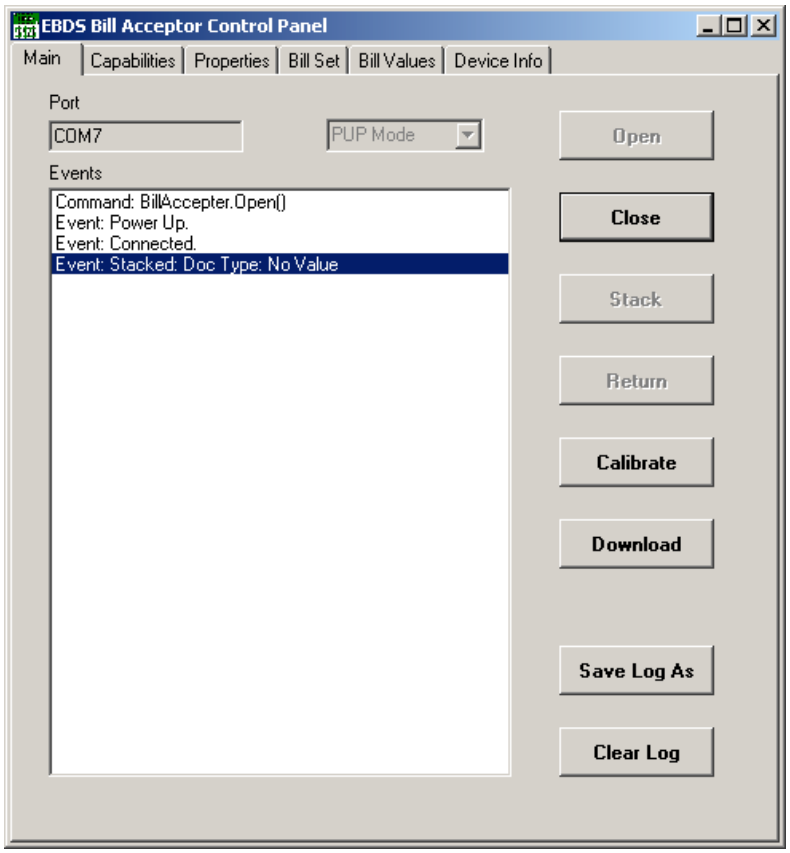
- The Open button opens a the connection to the bill acceptor.
- The Close button closes the control panel
- The PUP Mode combo-box is used to select the Power Up Protocol to be employed while opening the connection. See sections 6.1 and 7.1.1 for more details on the various PUP modes. PUP modes may only be selected BEFORE the connection to the device is opened. The combo-box is illustrated below:



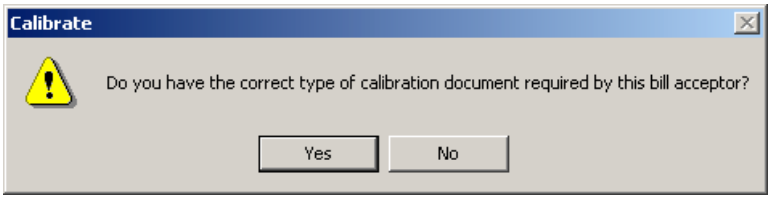
Once the connection to the device is established, the full range of options opens up.

14.2.1 The Main Tab

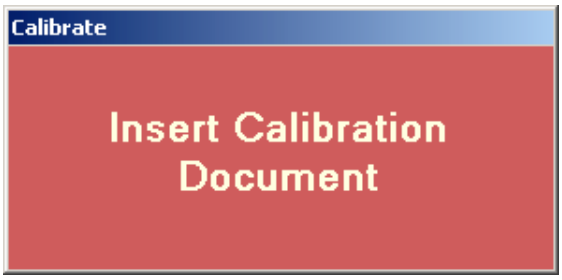
The main tab is used for the majority of bill operations. From this panel, the processing of documents may be seen. This panel is shown below:



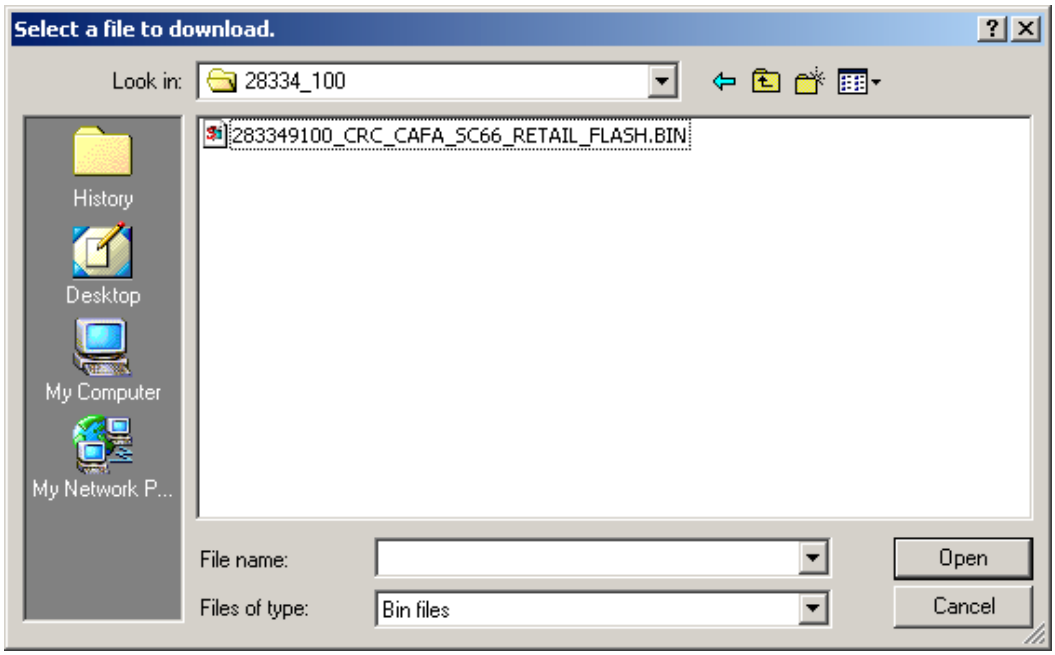
- The Events area is used to display commands sent to and events received from the bill acceptor object. The list of events scrolls as needed.
- The Save Log As button may be used to save the contents of the Events list in a text file.
- The Clear Log button erases the contents of the Events list.
- The Stack button is only enabled when there is a document in the bill acceptor awaiting a decision. If this button is clicked, the bill will be placed into the cash box.
- The Return button is only enabled when there is a document in the bill acceptor awaiting a decision. If this button is clicked, the bill will be returned to the consumer.
- The Calibrate button is used to initiate a field calibration procedure. During this procedure, a device specific calibration document is fed into the unit and analyzed to adjust the internal settings of the unit. Since it is crucial that the correct calibration document be used, the program then verifies this in the following pop-up:



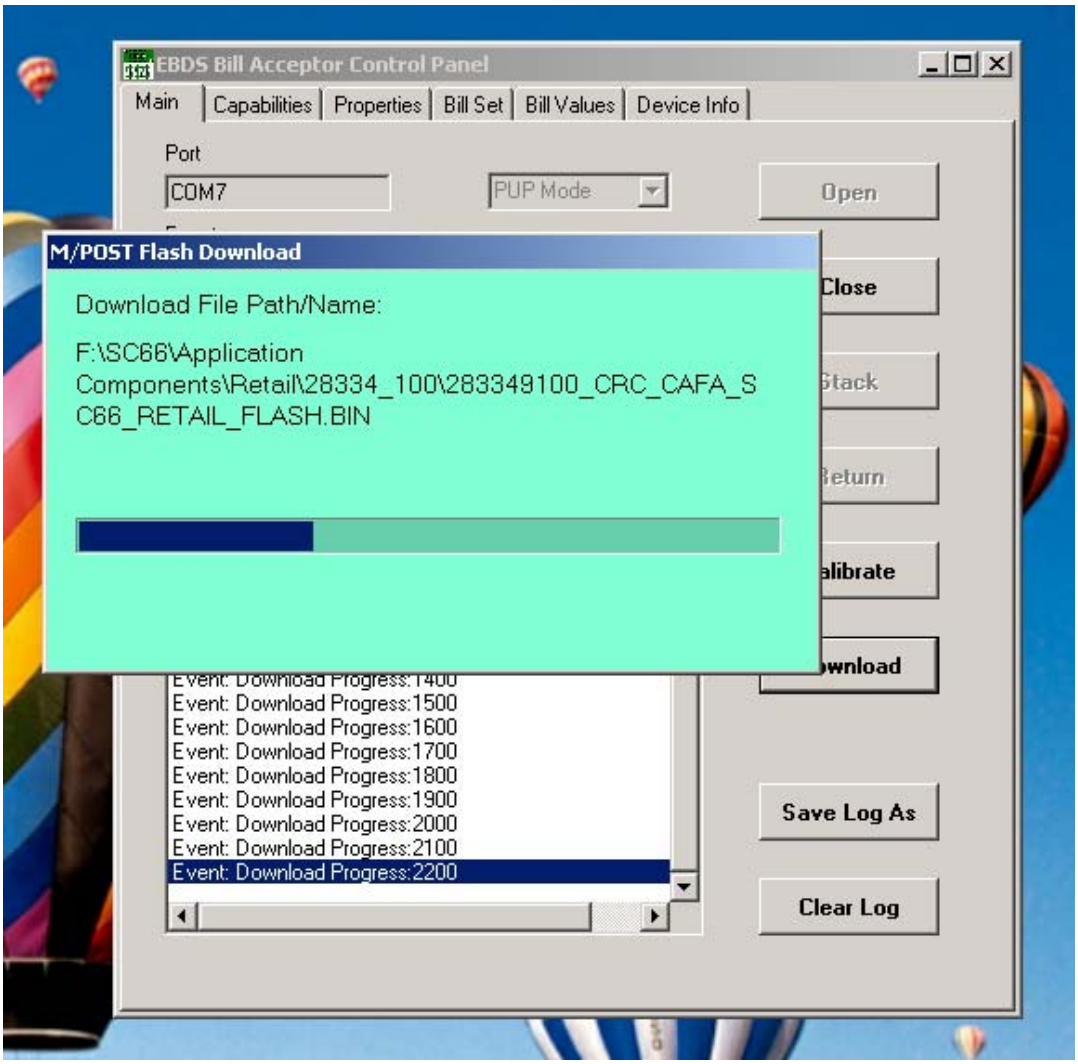
If YES is selected then the following dialog appears for the duration of the calibration process.



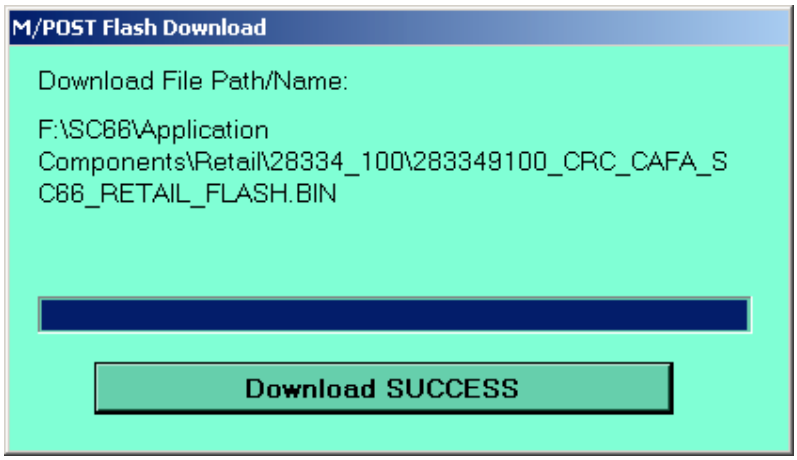
- The Download button is used to initiate an update of the code contained within the bill acceptor. When selected it first requires the selection of a file to load into the device. This is accomplished in the following dialog:



Once the file is selected, a progress dialog plots the progress of the exercise. This is shown below:



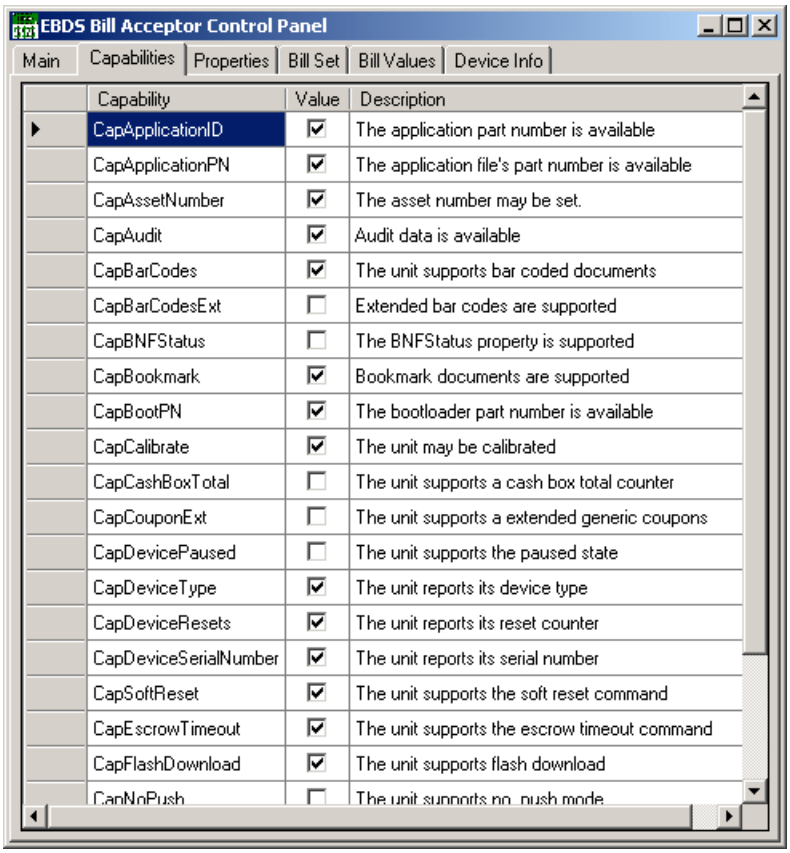
Note that download progress is tracked by both the download bar and the Event list in the main tab. When the download process is finally completed, the following is displayed:



Should the download fail, a download failed button is presented instead.

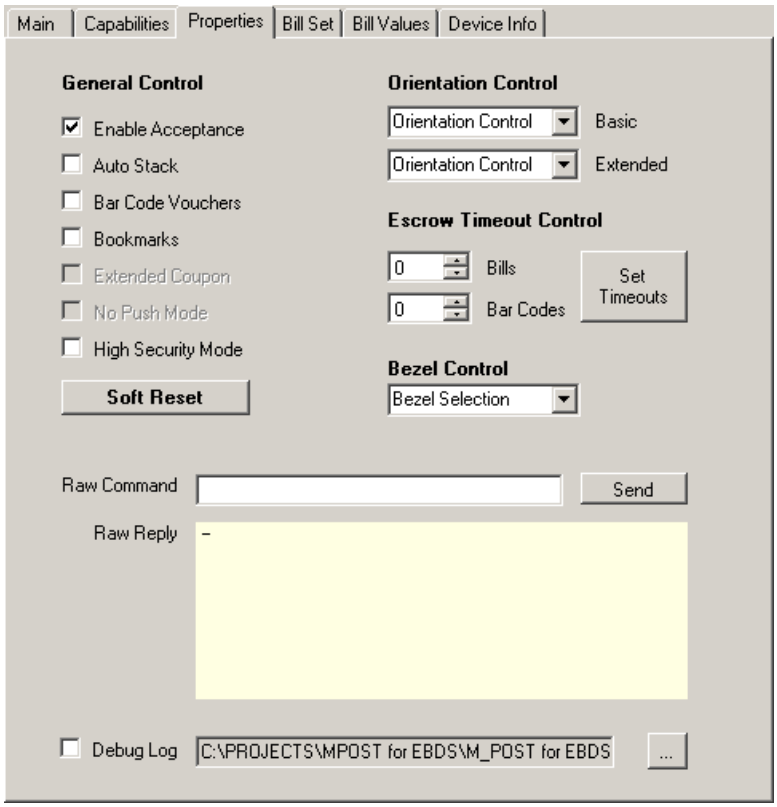
14.2.2 The Capabilities Tab

The capabilities tab is used to display the results of the M/POST capability survey of the bill acceptor. Each line in this table corresponds to a Boolean capability property described in sections 9.1.1 and 9.1.13. The value check box reflects the state of the capability. Checked being available and unchecked being not available. The value field is not editable. To help understand the nature of each capability, a description is provided as well. A sample capabilities table is shown below:



14.2.3 The Properties Tab

The properties tab is used to control the user modifiable properties of the bill acceptor. The properties are discussed further in sections 9.1.1 and 9.1.13. Properties that are “grayed out” are not supported by the current bill acceptor. A sample properties dialog is shown below:



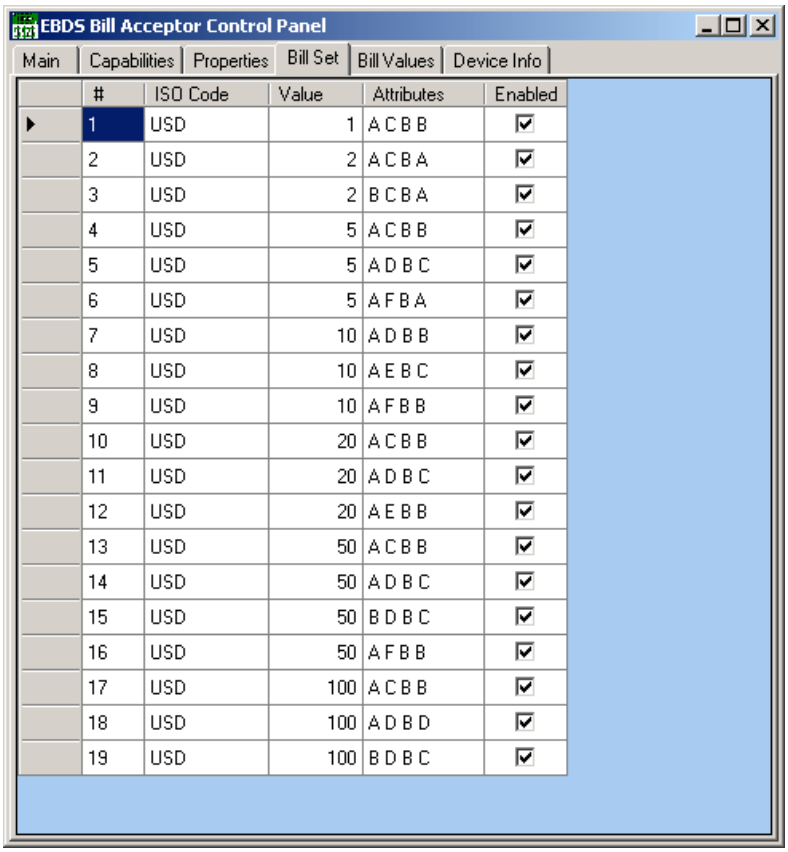
- The Enable Acceptance check box corresponds to the EnableAcceptance property. It controls the acceptance of all sorts of documents. Setting this property is the correct way to control device activity (abusing the open and close methods is the wrong way).
- The Auto Stack check box corresponds to the AutoStack property. When set, documents are stacked, bypassing the escrow event. When cleared, the escrow event is generated. In the demo, the disposition of the bill is then controlled by the Stack and Return buttons on the Main tab.
- The Bar Code Vouchers check box corresponds to the EnableBarCodes property. It controls the acceptance of bar coded vouchers.
- The Bookmarks check box corresponds to the EnableBookmarks property. It controls the acceptance of bookmark slips.
- The Extended Coupon check box corresponds to the EnableCouponExt property. When selected, generic coupons are reported in greater detail.
- The No Push Mode check box corresponds to the EnableNoPush property. When selected, the bill acceptor will go out of service rather than stack a bill to recover from a jam. Operator intervention will be required to clear the jam.
- The High Security Mode check box corresponds to the HighSecurity property. When set, stricter standards are used in the evaluation of documents. Note that while all bill acceptors accept this option, few act on it.
- The Soft Reset button is used to send a SoftReset command to the bill acceptor. This command is discussed in sections 9.1.10 and 9.1.13. There will a delay of several seconds while the bill acceptor performs this action.
- The Orientation Control area is used to control the orientation of accepted notes. The first orientation control combo-box corresponds to the basic orientation control capability. This is

described in section 8.2.4. The second orientation control combo-box corresponds to the extended orientation control capability. This is described in section 8.2.5.

- The escrow timeouts for bills and barcode documents may be set in the Escrow Timeout Control area. The timeouts for bills and barcodes are dialed in with the appropriate spin boxes and the setting sent to the bill acceptor with the Set Timeouts button.
- The Bezel Control area is used to override the bill acceptor bezel setting.
- The Raw Command area allows arbitrary hex values to be entered and sent to the bill acceptor. The data is entered with the STX, Length, ETX and Check value omitted. To send the command, either hit Enter while the focus is in the command area or click on the Send button. The Raw Reply area shows the complete reply packet.
- The Debug Log check box corresponds to the DebugLog property. When enabled, a log of all traffic is written to the folder listed in the text area to the right. The text area corresponds to the DebugLogPath property. The button labeled "..." to the right of the text area is used to select a folder to place the log in. Note that selecting a new folder while the log is active has no effect. The folder must be selected *before* the log is started. It is allowed to start logging on a connection that is not yet open. When the connection is opened, logging will begin immediately. Be advised that logging can consume a great deal of system resources and disk space and should be used with care.

14.2.4 The Bill Set Tab

The bill set tab is used to display the complete bill set supported by the bill acceptor and corresponds to the BillTypes and BillTypeEnables properties. The enable check boxes permit fine control over exactly what sorts of bills are accepted. An example of the bill types tab is shown below:

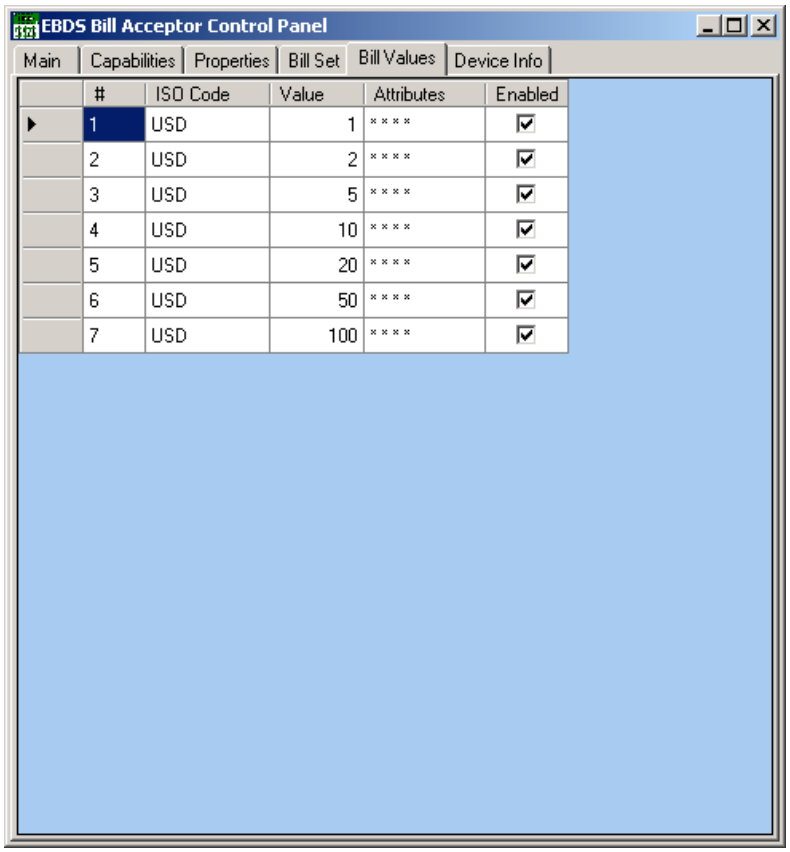


#	ISO Code	Value	Attributes	Enabled
1	USD	1	A C B B	☑
2	USD	2	A C B A	☑
3	USD	2	B C B A	☑
4	USD	5	A C B B	☑
5	USD	5	A D B C	☑
6	USD	5	A F B A	☑
7	USD	10	A D B B	☑
8	USD	10	A E B C	☑
9	USD	10	A F B B	☑
10	USD	20	A C B B	☑
11	USD	20	A D B C	☑
12	USD	20	A E B B	☑
13	USD	50	A C B B	☑
14	USD	50	A D B C	☑
15	USD	50	B D B C	☑
16	USD	50	A F B B	☑
17	USD	100	A C B B	☑
18	USD	100	A D B D	☑
19	USD	100	B D B C	☑

14.2.5 The Bill Values Tab

The bill set tab is used to display a summarized version of the bill set supported by the bill acceptor and corresponds to the BillValues and BillValueEnables properties. The enable check boxes permit easy control over exactly what denominations of bills are accepted.

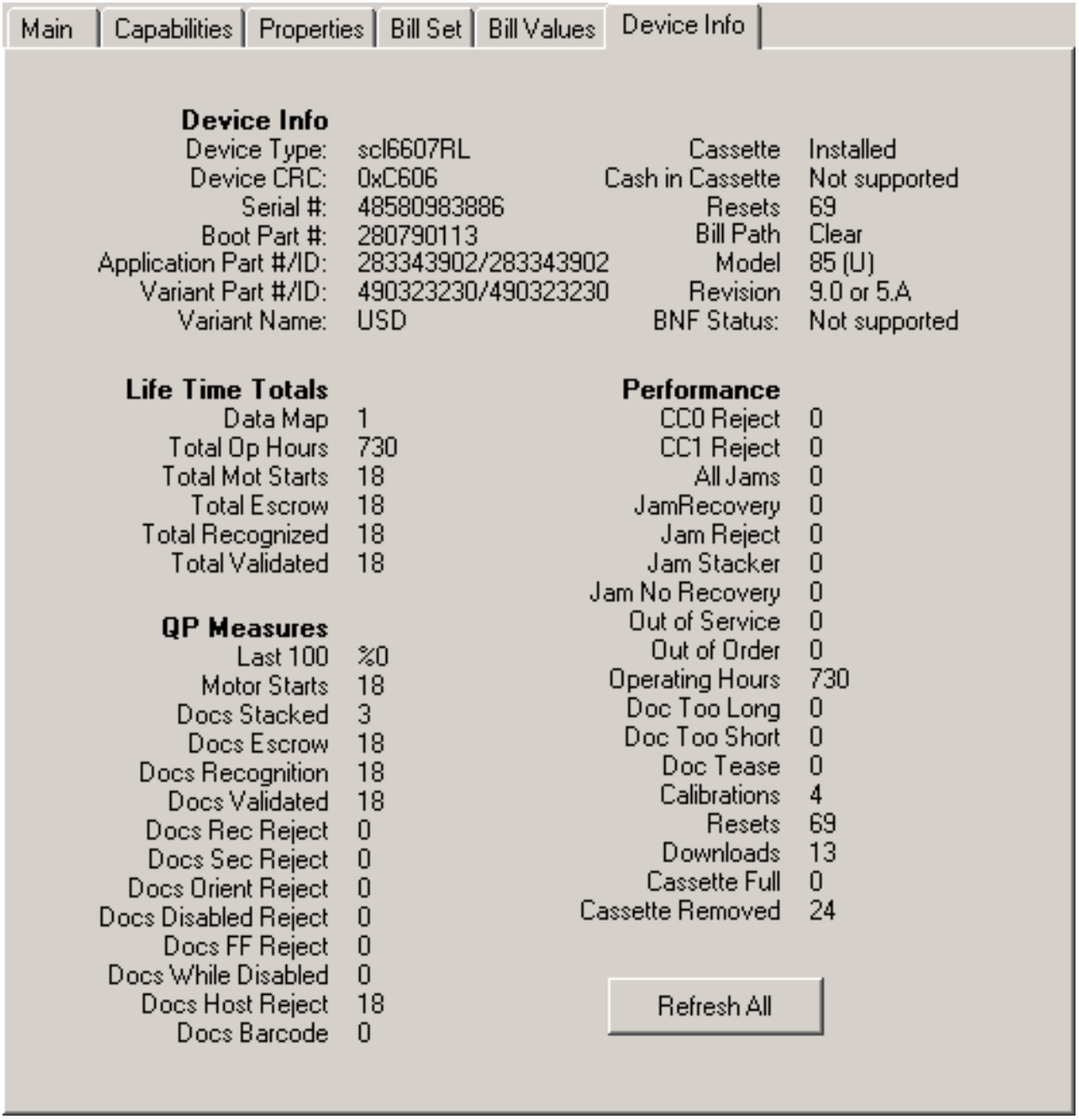
Note that while changes in this tab are reflected in the Bill Types tab, changes in the Bill Types tab do *not* reflect in this tab. Thus it is recommended that only one tab be used to control the acceptance of bills. An example of the bill values tab is shown below:



Note: On bill acceptors that do not support expanded note reporting, the bill types tab and bill values tab contain the same data.

14.2.6 The Device Info Tab

The last tab is the device info tab. This tab contains a great deal of detailed information about the bill acceptor. These various fields are populated according to the capability of the device to report the requested information. Where fields are not available the text “Not Supported” appears instead of the data. The refresh all button updates this tab with fresh information. A sample of the device info tab appears below:



15. Device Harness and Connection

15.1 EBDS Harness Options for the Series 2000 Bill Acceptor:

There are two main harness options for the Series 2000 Bill Acceptor. These are the RS232 harness and the USB harness. These options may be

250078075P – RS232 SERIES 2000 INTERFACE CABLE
250079066P1 – SERIES 2000 USB INTERFACE HARNESS

14.1.1 Custom Harness Design for the Series 2000 Bill Acceptor:

In some cases, it may be desired to avoid intermediary cables or harnesses. In these cases a custom cable must be fabricated to interface directly to the Series 2000 unit. The specifications for this connection are shown below:

The Series 2000, 30 pin connector

12 pin power sub-connector						**18 pin communications sub-connector**								
									T	**A**	**B**			
1	2	**3**	**4**	**5**	**6**	7	8	9	**10**	**11**	12	13	14	15
16	17	18	**19**	**20**	**21**	22	23	24	**25**	**26**	27	28	29	30

EBDS Data Connections:

Pin 10 – Signal Ground.
Pin 11 – Inverted TTL Serial Data to the Host.
Pin 25 – +5 V DC through a 200 ohm resister.
Pin 26 – Inverted TTL Serial Data from the Host.

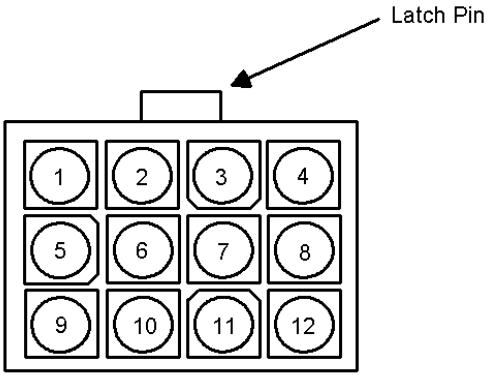
Power sub-connector pin	**Type of Bill Acceptor**		
	115V AC	**24V AC/DC**	**12V DC**
3		24V Hot	
4	115 VAC Neutral		
5		Key	Key
6	Key		
16			12 VDC "–" Neutral
19	Key		12 VDC "+" Hot
20	115 VAC Hot	24V Neutral	
21	Earth Ground	Key	Key

15.2 EBDS Harness Options for the Cashflow-SC Bill Acceptor:

The various models of the Cashflow-SC support EBDS through a number of interfaces. These are controlled by the model number of the unit ordered. For ordering information, contact MEI Sales directly.

14.2.1 RS-232 Harness Configuration:

The RS-232 versions of the Cashflow-SC employ a 12 pin Molex style connector shown below.



Mating Connector: Housing - Amp #172333-1
Pins - Amp #170360-1 or #170364-1

12 Pin Chassis Docking Station Connector (End View)

The pin-outs of this cable are specified below:

CASHFLOW SC 12 Pin Block Connector Pin-out for RS232 EBDS version

Connector Pin #	Wire Color	Signal
1	White	$\overline{\text{External Inhibit}}$
2	Gray	$\overline{\text{Bezel LED Drive}}$
3	Not Populated	
4	Yellow	$\overline{\text{Out of Service}}$
5	Blue	Ground [2]
6	Pink	RS232 EBDS RXD[1]
7	Blue	Power Supply Return [2]
8	Purple	Led Supply
9	Not Populated	
10	Not Populated	
11	Green	Power Supply [3]
12	Tan	RS232 EBDS TXD[1]

NOTES: [1] RXD is an input to the note acceptor. TXD is an output. All signals are true RS-232 levels.
[2] Pins 7 and 5 are tied with a loop of wire in back of the 12-pin connector.
[3] The power supply should be rated for 24 volts and 72 watts over the entire temperature range of the system.

15.2.2 USB Harness Configuration:

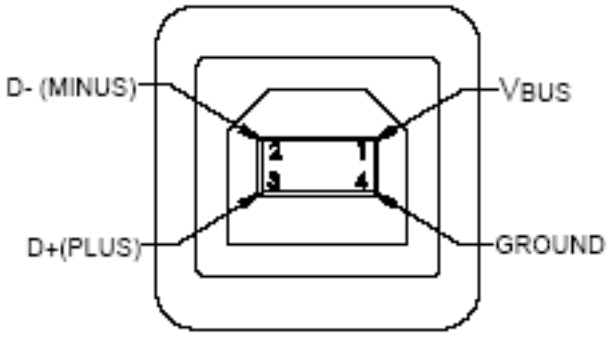
The USB versions of the Cashflow-SC utilize two harnesses. The first is the conventional 12 pin Molex connection and the second is a USB "B" type connector for data communications with the host system.

CASHFLOW SC 12 Pin Block Connector Pin-out for USB EBDS version

Connector Pin #	Wire Color	Signal
1	White	$\overline{\text{External Inhibit}}$
2	Gray	$\overline{\text{Bezel LED Drive}}$
3	Not Populated	
4	Yellow	$\overline{\text{Out of Service}}$
5	Blue	Ground[1]
6	Not Populated	
7	Blue	Power Supply Return[1]
8	Purple	Led Supply
9	Not Populated	
10	Not Populated	
11	Green	Power Supply [2]
12	Not Populated	

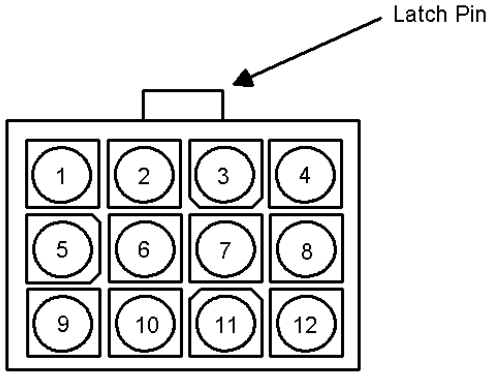
NOTES: [1] Pins 7 and 5 are tied with a loop of wire in back of the 12-pin connector.
[2] The power supply should be rated for 24 volts and 72 watts over the entire temperature range of the system.

CASHFLOW SC Pin-out for USB "B" Socket.



15.3 Legacy, Series (ZT) 1000 Harness Options

The series 1000 harness is a 12 pin connector similar to that used on the Cashflow-SC product line. The following table summarizes EBDS harness options for the Series 1000 bill acceptor product line. This data is provided for informational purposes only.



Mating Connector: Housing - Amp #172333-1
Pins - Amp #170360-1 or #170364-1

12 Pin Chassis Docking Station Connector (End View)

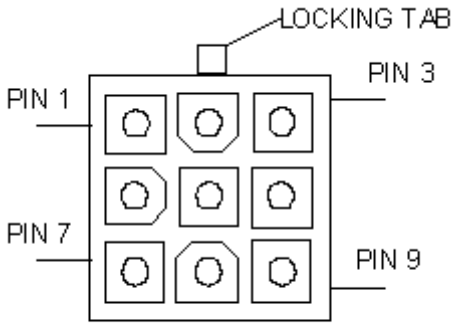
Connector Pin #	Wire Color	INTERFACE SIGNALS		
		RTU Harness # 251076029 Yellow Jacket	RTU Harness # 251072043 Red Jacket	RTU Harness # 251071030 Blue Jacket
		Product - ZT1207	Products – ZT1202, ZT1204	Product - ZT1201
		Interfaces: NISR / RS232 EBDS	Interfaces: IGT® Netplex, IGT® Pulse, Opto-isolated EBDS	Interface: Opto-isolated EBDS
1	White	CASSETTE PRESENT	Aux. A	----
2	Gray	BEZ_LED_OUT	LED-	Aux. B
3	Red	NISR_SEND	Vopt	Vopt
4	Yellow	OUT_OF_SERVICE	Vret	Vret
5	Blue	GROUND	GROUND	GROUND
6	Pink	RS232 RXD	Isolated Reset	Isolated Reset
7	Black	NISR_INTERRUPT	Aux. B	----
8	Purple	LED_SUPPLY	LED+	Aux. C
9	Brown	TXD/CREDIT	TXD	TXD
10	Orange	OPT_RXD/ACC_EN	RXD	RXD
11	Green	POWER	POWER	POWER
12	Tan	RS232 TXD	----	Aux. A

15.4 Legacy, Series 3000 Harness Options

There is one main harness option for the Series 3000 Bill Acceptor. This is a RS232 harness:

111633139 – SERIES 3000 RS232 INTERFACE BOARD KIT

Direct connection, without the use of a harness is not recommended or supported. Power to the Series 3000 unit is provided by a nine-pin harness. This harness and its configuration are shown below:



9 PIN POWER CONNECTOR COMMING FROM THE ACCEPTOR

Power connector pin	Type of Bill Acceptor	
	115V AC	24V AC
4	115 VAC Hot	
5		24VAC Hot
6	115 VAC Neutral	24 VAC Neutral

This page intentionally left blank.

16. Quick Reference

Controller's Message

STX (0x02)

Length (0x08)

Description		"1" Indicates
Message Type / Ack		
Bit 0	Message Sequence	Toggles
Bit 1-3	Reserved – Always 0	
Bit 4-6	Message Type (See Below)	
	High Nibble	Type
	0x	Reserved
	1x	Standard Host Command
	2x	Standard Device Reply
	3x	Host Command + Bookmark
	4x	Calibrate Request
	5x	Flash Download Request
	6x	Auxiliary Commands
	7x	Extended Message Set

Byte 0		
Bit 0	Denom 1 Accept Enable	Enable
Bit 1	Denom 2 Accept Enable	Enable
Bit 2	Denom 3 Accept Enable	Enable
Bit 3	Denom 4 Accept Enable	Enable
Bit 4	Denom 5 Accept Enable	Enable
Bit 5	Denom 6 Accept Enable	Enable
Bit 6	Denom 7 Accept Enable	Enable

Byte 1				
Bit 0	Special Interrupt Mode			Mode Enabled
Bit 1	High Security			Enabled
Bit 2-3	Bill Orientation Enable (See table below)			
	Bit	3	2	Orientation
		0	0	1 Way
		0	1	2 Way
		1	x	4 Way
Bit 4	Escrow Mode			Escrow Enabled
Bit 5	Stack Command Bit			Stack the Bill
Bit 6	Return Command Bit			Return the Bill

Byte 2		
Bit 0	Push/No Push Modes	No Push Mode
Bit 1	Decode Bar Codes	Enable
Bit 2	Power Up B Sequence	Enable
Bit 3	Power Up C Sequence	Enable
Bit 4	Expanded Bill Set	Enable
Bit 5	Expanded Coupon	Enable
Bit 6	Currently 0	RFU

ETX (0x03)

CheckSum

Acceptor's Message

STX (0x02)

Length (0x0B)

Message Type / Ack (see controller description)

Description		"1" Indicates
Byte 0		
Bit 0	Idling	Waiting for a Bill.
Bit 1	Accepting	Taking a Bill
Bit 2	Escrowed	Bill is in Escrow
Bit 3	Stacking	Stacker is moving
Bit 4	Stacked	Bill was Stacked
Bit 5	Returning	Returning a Bill
Bit 6	Returned	Bill was Returned

Byte 1		
Bit 0	Cheated	A cheat was detected
Bit 1	Rejected	Bill was Rejected
Bit 2	Jammed	Bill is Jammed
Bit 3	Stacker Full	Stacker is Full
Bit 4	LRC Status	LRC Installed
Bit 5	Paused	Acceptor is Paused
Bit 6	Calibration	Acceptor is Calibrating

Byte 2					
Bit 0	Power Up				The device was reset
Bit 1	Invalid Command				Bad command from host
Bit 2	Failure				Out of Service
Bit 3-5	Bit	5	4	3	Bill Value / Denom
	0x00	0	0	0	None/No value
	0x08	0	0	1	Denom 1
	0x10	0	1	0	Denom 2
	0x18	0	1	1	Denom 3
	0x20	1	0	0	Denom 4
	0x28	1	0	1	Denom 5
	0x30	1	1	0	Denom 6
	0x38	1	1	1	Denom 7
Bit 6	Currently 0				RFU

Byte 3		
Bit 0	Push/No Push	Acceptor is stalled.
Bit 1	Flash Download	Starting Flash D/L.
Bit 2	Pre-stack	Obsolete
Bit 3	Raw barcode	Supports 24 byte codes
Bit 4	Device Caps	Allows QryDeviceCaps
Bit 5-6	Currently 0	RFU

Byte 4	
All	Model # (00-7FH)

Byte 5	
All	Code Revision (00-7FH)

ETX (0x03)

CheckSum

Warning
Any RFU fields may be changed at any time.

17. Hex/Binary and ASCII Data Conversion

When developing a host in an higher level language, one is often isolated from the low level "bits" of the protocol. The following table facilitates the decoding of the seven bit data contained in an EBDS data stream. The low and high hex digits may be translated into the appropriate bit positions. This in turn facilitates looking up fields in the various sections of this specification.

Digit	Hex→	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Low	Bit 0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	Bit 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	Bit 2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	Bit 3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
High	Bit 4	0	1	0	1	0	1	0	1								
	Bit 5	0	0	1	1	0	0	1	1								
	Bit 6	0	0	0	0	1	1	1	1								

Since many EBDS packets contain ASCII data, this Hex to ASCII conversion table is provided to ease the interpretation of transaction logs.

H/L	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*	*BS*	*TAB*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
1	*DLE*	*DC1*	*DC2*	*DC3*	*DC4*	*NAK*	*SYN*	*ETB*	*CAN*	*EM*	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	*DEL*

Notes: Entry 20 is a space character. Entries in *italics* are non-printable control codes.