

Milan / Paylink System Manual

Aardvark Embedded Solutions Ltd does not accept liability for any errors or omissions contained within this document. Aardvark Embedded Solutions Ltd shall not incur any penalties arising out of the adherence to, interpretation of, or reliance on, this document. Aardvark Embedded Solutions Ltd will provide full support for this product when used as described within this document. Use in applications not covered or outside the scope of this document may not be supported. Aardvark Embedded Solutions Ltd. reserves the right to amend, improve or change the product referred to within this document or the document itself at any time.

Table of Contents

Table of Contents	2
Revision History	6
Introduction	7
Purpose of Document	7
Intended Audience	7
Associated Document(s)	7
Naming	7
Supported Facilities	8
Version Numbering	8
Document Structure	8
Paylink	9
Installation.....	9
Money representation	9
Acceptance	9
Payment.....	9
<i>Payout Function</i>	9
<i>PaySpecific Function (1.12.6)</i>	10
<i>Processing during payout</i>	10
<i>End of payout processing</i>	11
Auxiliary Items.....	11
<i>Switch inputs</i>	11
<i>Outputs</i>	11
<i>Meters</i>	12
System Structure	12
<i>USB Connection</i>	12
Paylink Lite.....	13
<i>Original (old) Paylink Lite</i>	13
Paylink Lite V2 ccTalk.....	13
Paylink MDB Lite.....	13
Paylink Lite V2 RS232	13
Paylink Lite Aux units.....	14
User control of driver program	14
Troubleshooting	15
Supported Peripherals	16
Coin / Note Acceptor Usage Details	18
Token Handling (Coin Ids) (1.11.x)	18
Dual Currency Handling (Coin Ids) (1.11.x)	18
Coin Routing.	19
<i>Route coins to a general cash box</i>	19
<i>Route specific coins to a specific cash box</i>	19
<i>Route coins to a hopper until it is full then route it to a coin cash box</i>	19
<i>Paylink Routing - Flow Diagram</i>	20
Control of Motorised Acceptors.....	21
<i>ccTalk bulk coin acceptor (1.11.3)</i>	21
<i>BCR / CR10x coin recyclers (1.11.5)</i>	21
MDB changer / BCR / CR10x / CLS recycler / SmartHopper support.	22
<i>MDB Payout</i>	22

<i>MDB tube level monitoring</i>	23
Read out of Acceptor Details (1.11.x)	24
Coin / Note Dispenser Usage Details	25
Dispenser Power Fail support.	25
Detailed Device Support.	25
<i>Abandoning a payout in progress (1.11.3)</i>	25
<i>Control of unwanted bill payout (1.11.3)</i>	25
Combi Hopper Support.	25
Read out of Dispenser Details (1.11.x)	26
Complex Dispenser (Recycler) Operations (1.12.3)	27
Introduction	27
Security	27
<i>DES Key Exchange</i>	27
Component Identity	28
Routing	28
<i>Dispenser Destination Initialisation</i>	28
<i>Routing Control</i>	28
Bill Recycler Emptying	29
<i>Full Dump</i>	29
<i>Partial Dump</i>	29
Payout Progress	31
<i>Cancelling Payout</i>	31
<i>Notification of progress</i>	31
Power Fail	31
<i>Temporary power interruption</i>	31
<i>Full power Failure</i>	31
<i>Unpaid Bills</i>	32
Device Specific Functionality	33
<i>MEI BNR</i>	33
<i>SCR Advance (EBDS)</i>	33
<i>SCR Advance Mixed Denomination Recycler (1.12.12)</i>	34
<i>JCM Vega (cctalk DES) & Innovative NV11 Recycler (DES)</i>	34
<i>ICT BR2300 (Not available with DES)</i>	35
<i>MDB Note recycler</i>	35
<i>Innovative NV200 Recycler / SmartPayout (DES)</i>	36
<i>Innovative SmartHopper coin recycler</i>	36
<i>JCM UBA & iPro Recycler</i>	36
<i>F56 / F53 Bill Dispenser</i>	37
<i>F56 / F53 Jams</i>	37
<i>Cashcode B2B-300</i>	37
<i>Cashcode B2B-60</i>	38
<i>Merkur 100</i>	38
Extended Escrow (1.12.6)	39
Introduction	39
Functionality	39
<i>Accepting Notes</i>	40
<i>Returning Notes</i>	40
<i>Keeping Notes</i>	40
Operation	41
Abnormal Situations	42
Cashless Processing	43
Background	43
Processing	44

<i>Credit Card Sequencing</i>	44
<i>Credit Input Sequencing</i>	44
<i>Credit Output Sequencing</i>	44
<i>Ticket Sequencing</i>	45
Abnormal Processing	45
MDB Cashless processing	45
<i>MDB Cashless Credit</i>	46
Example Cashless Transaction	47
Meters / Counters	48
<i>Mechanical Meters (1.12.4)</i>	48
Events (Faults / Auditing)	49
Introduction	49
cctalk coin processing	50
<i>Fault Events</i>	50
<i>Coin Events</i>	50
cctalk note processing	51
<i>Fault Events</i>	51
<i>Note Events</i>	51
cctalk hopper processing	52
ID-003 note processing	54
<i>Fault Events</i>	54
CCNet note processing	55
<i>Fault Events</i>	55
EBDS (SC/SCR) note processing	56
<i>Fault Events</i>	56
BCR / CR10x Fault Processing	57
CLS Fault Processing	60
F53/F56 Fault Processing	60
Firmware reprogramming	61
Command Line Options	62
Limitations	62
Milan / Paylink Driver Program Configuration	63
Driver Parameters	63
Multiple Paylink Unit Support	64
<i>Unit Identification</i>	64
Operating modes	64
External Paylink Peripheral Specification	65
The Configuration File	66
DRIVER Details	67
SYSTEM Details	68
PROTOCOL Details	69
CCTALK Device Definition	70
CCNet Device Definition	73
MDB Device Definition	74
ID003 Protocol	74
EBDS Protocol	75
MEIBNR Protocol	75
TFLEX T-FLEX Protocol	75
CX25 Protocol	76
CLS Protocol	76

Gen2 Protocol76
F56 Protocol.....76
MFS Protocol76
Cassettes.....76
Original Paylink Definition.....77
Disclaimer.....78

Revision History

Version	Date	Author	Description
1.0	25 th Feb 13	D Bush	Created from "Configurable Driver" Manual
1.1	24 June 2013	D Bush	Update for Precise pay and Extended Escrow
1.2	1 August 2013	D Bush	Updated for SmartHopper support
1.3	12 December 2016	D Bush	1.12.8 updates
1.4	11 September 2019	D Bush	1.12.9 & 1.12.10 updates
1.5	29 January 2020	D Bush	1.12.11 updates
1.6	16 March 2020	D Bush	Added SCR Mixed Denomination Recycler details
1.7	16 March 2022	D Bush	Cashless and General updates
1.8	16 April 2024	D Bush	MDB and auxiliary Lite and general updates
1.9	12 November 2024	D Bush	Added SmartCoin config option

Introduction

Purpose of Document

This document describes the structure of a system using the AES Intelligent Money Handling Equipment Interface (Milan / Paylink), as seen by the person designing and setting up the system

Intended Audience

The intended audience of this document is the system engineer or programmer who is configuring the system that will be using Paylink.

Associated Document(s)

This document is one of a pair that together cover creating and using a Paylink system. This document is written for the use of the person who is possibly not a programmer, but is concerned with designing and setting up the system centred on a Paylink unit. That document covers and the way in which such units can be controlled and the configuration settings that are used to configure the system.

The companion document “Milan / Paylink Application Program Interface Manual” for the use of programmers and covers the details of how to write the programs that interface to Paylink.

Naming

The system described here has a few names. This section attempts to explain them.

AES	Aardvark Embedded Solutions - us.	
IMHEI	Intelligent Money Handling Interface Equipment. This was the original name for the project, This was however difficult to say, and so was replaced in common use by Milan. It remains in the names in of the header files etc.	
Milan	This was originally the name of the first hardware build. It has however become the name of the overall project. Most documents from AES talk about Milan to cover the whole family of products that are used with this API	
Paylink	This is the name of the USB module. There are at present seven versions of Paylink hardware	
	Standard Paylink	The original, metal cased version.
	Paylink Lite	A old, smaller, plastic cased cctalk only version, with a reduced function set.
	uPaylink	(Micro Paylink) a PC software only version, for use with Crane PI USB peripherals.
	Paylink Lite V2	A smaller, plastic cased version supporting the full set of peripherals on a single connector, either cctalk or RS232 together with 4 input and 4 outputs
	Paylink MDB Lite	A smaller, plastic cased version supporting the full set of peripherals on a single MDB connector together with 4 input and 4 outputs
	Paylink MDB Aux	A version of the MDB Lite with no I/O that does not authorise Paylink to run
	Paylink RS232 Aux	A specially programmer USB to RS232 converter that does not authorise Paylink to run

Supported Facilities

It should be noted that this document cover all versions of the Milan / Paylink system.

Where a facility may not be available with the version that you are running, the topic titles are suffixed with a version indication in brackets

Version Numbering

All AES software releases have a 4 part version number. This is made up from 4 separate fields, coded as:

L L - P P - V V - M M

where:

- M M Is a minor release, representing an upgrade in facilities or bug clearance, but where the application code will remain the same (both source and executable).
- V V is a significant release, where the application will at a minimum need to be re-compiled, and where facilities may have changed to the point where the application code needs to change.
- P P Is a product code. This is 1 for Paylink and is 25 for DES Paylink
- L L Is the release level. This is a code, rather than a level, and has meaning as follows:
- 4 is a full release, and should never contain any errors or omissions. These releases happen relatively rarely and a full history of the code is maintained. A code starting 4 uniquely identifies a particular build of the software.
 - 3 is a beta release. This may contain errors as it has not been fully regression tested, but it is intended to be sufficiently stable that development, or even live running is possible. Again, a code starting 3 uniquely identifies a particular build of the software. Normally the full release of a version will be almost identical to the beta release.
 - 2 is an alpha release. These are only usually issued at the start of a major version. They *should* be stable and bug free, but are not fully tested, especially they will only have had minor regression testing. This release is to enable developers to “get started” with a new set of facilities. Again, a code starting 2 uniquely identifies a particular build of the software.
 - 1 is an engineering release. These are generated during our internal development process, and are occasionally released to customers in response to specific requests. A build code of 1 can only be distinguished by the date / time stamp embedded in the code, and no internal record is kept of the items / changes that have gone into such a build.
- Note: These are always part of the main line development, any new functionality they support is guaranteed to part of the full release with the same VV & MM.
-

Document Structure

This document is divided into three overall parts:

Concepts

Where the document describes the ideas behind how Paylink works

Details

Where the specifics of how Paylink handles peripherals and situations are described

Configuration

Which defines the configuration file (which is essential to Paylink operation).

Paylink

Installation

All aspects of actually installing Paylink software on a target PC are described in the companion Application Program Interface document.

Money representation

Within Paylink all monetary figures are in 32 bit integers, which represent an amount of money in terms of a single base unit. This would typically be pence or cents, but could be yen etc.

Where note acceptors are reading in high value notes, the acceptor will typically provide a conversion factor, which enables Paylink to convert the notes. A “normal” dollar / euro acceptor will provide a conversion factor of 100.

Acceptance

All money acceptance is handled by means of updating total counters. Before starting operation, the application notes the current value of all the counters in which it is interested, and then monitors these counters for changes.

This serves to remove all needs for queuing and for spotting events from the system - there is no way that application can fail to have accurate information.

For the simplest application, there is a single total of all credit received. This actually totals the credit received for the life of the unit, and hence can also be used for auditing / security purposes.

For a more complex understanding of the money received, Paylink provides a block of information for each acceptor. As well as being able to use this block to disable specific coins / notes it also monitors the insertion of each coin / note. For each coin / note the total number accepted since the Paylink unit was reset is reported.

Payment

Paylink provides two similar mechanisms for paying currency out from dispensers to the users of the system.

The original system used the **Payout()** function and with this the application specified the total amount, and Paylink would attempt to pay out sufficient of the available notes and coins to total the specified amount.

The new (1.12.6) precise pay system uses the **SetDispenseQuantity()** and **PaySpecific()** functions to pay out a precisely specified set of notes and coins.

Payout Function

This method of paying money out using a Paylink is by calling the **Payout()** function, which takes a value in Paylink base units.

Paylink maintains a count of the total of all credit paid out for the life of the unit. This total is updated continuously as the process of paying money out proceeds, and can be used to check the amount of credit that has been paid out in the event of a payout being in progress when power is lost.

A Paylink is connected to one or more *dispenser* devices, which can be used to achieve this payout. Internally Paylink holds these devices in descending order of value and when a pay command is issued it works down this list, paying as many base units as possible from each device in turn.

Each device request will be successful, or will result in nothing being paid, or will pay less than the requested amount.

Where either nothing or less than the requested amount is paid then Paylink will automatically issue another request on that device for the remainder. When two successive requests have resulted in nothing being paid, then Paylink abandons the use of that device for this command, and will attempt to pay the outstanding balance from lower value.

The application can exert limited control over progress of a payment by disabling specific dispensers, which has the effect of causing the dispenser to be ignored when selecting which units to use for a payment.

PaySpecific Function (1.12.6)

The alternative method of paying money out using a Paylink is by calling the **SetDispenseQuantity()** function, once for each dispenser that is to be used for the payout to specify how many coins / notes are required from that..

When all the required calls have been made, a single call to the **PaySpecific()** function will Paylink to start processing the payout.

The call to the **PaySpecific()** function, will return the total amount to be paid in Paylink base units, in many ways subsequent processing is the same as a that triggered by a **Payout()** call for this value.

Internally Paylink holds these devices in descending order of value and it works down this list, paying as many base units as possible from each device in turn.

Each device request will be issued and the result used to update the Status of the relevant Dispenser.

If the application disables specific dispensers, this will still have the effect of causing the dispenser to be ignored when it is reached in this processing.

Processing during payout.

As money is paid out during either process, Paylink updates the total of all credit paid out as reported by **CurrentPaid()**.

Specifically, this value will increment as coins / notes are delivered, even though the **LastPayStatus()** is still reporting PAY_ONGOING. For a note dispenser this value increments as soon as the note is accessible, even though the payout doesn't terminate until it has been removed.

Whilst either payment command is being processed, the status of the system as reported by **LastPayStatus()** is PAY_ONGOING - when the entire payout process is complete it will either report PAY_FINISHED (indicating the value request has been paid out) or it will report the last failure code that was processed.

End of payout processing.

You detect the process completing using LastPayStatus(). If Paylink succeeded in paying out the total requested amount, you get a status of PAY_FINISHED. If not then Paylink will report the status returned by one of the dispensers that failed.

The amount actually paid out can be obtained by comparing the current lifetime credit from CurrentPaid() with that immediately before the payout started, the increment in will reflect the amount delivered.

The application can read the details on each dispenser from Paylink. The information available includes the value of the money in the dispenser device, the lifetime count of money paid from this dispenser as retrieved from the hardware unit, the result of the last attempt to pay out *using this dispenser* and if available the count of the units of money currently in the dispenser. It is possible that a successful payout still results in an individual dispenser returning an error status.

The values returned the DISPENSER_BLOCK are as follows:

Status - this is usually the result of the last payout attempt - even if that was a "long" time ago. It also can return PAY_US if the unit is no longer connected.

Count - increases in this value match the number of notes that have been delivered / taken.

CoinCount - decrements in this match the number of notes that have been delivered or stacked or "gone missing".

Where Paylink is not actively running the payout process when a note is delivered, this can be detected by an increment in Count (as above). This value is preserved over Paylink restarts / resets so can be used by the application to spot this. In addition an event of IMHEI_NOTE_DISPENSER_UPDATE is queued to the event system.

For note recyclers, notes no longer in the recycler unit can be detected by a decrement in the value of CoinCount, again this value is preserved over Paylink restarts / resets - a decrement of this without a corresponding increment in Count represents a note that has gone to the cashbox.

Auxiliary Items

Switch inputs

Standard Paylink provides 16 "open collector" style inputs, each of which has a pull up resistor to 3V3 and can discriminate between an open input, and one that has been grounded using a switch or transistor.

These are monitored and "debounced" on a millisecond timeframe, and two counts are made available to the application for each switch, one for the number of close, and one for the number of opens. Both counts are zero when the unit is reset - an application can monitor the count of closes by looking at one or the other, and can determine the current state of the switch by testing if they are equal.

Paylink Lite V2 and Paylink MDB Lite provide 4 inputs that work in an identical fashion.

Outputs

Standard Paylink provides 16 "open collector" style output, each of which consists of a transistor which can be operated under application control to connect an external load to ground. The Standard Paylink board provides an easily accessible source to enable LEDs to be easily connected to these outputs.

Paylink Lite V2 and Paylink MDB Lite provide 4 outputs that work in an identical fashion.

Meters

The final piece of equipment support by Paylink is external meters. Two sorts of meters are supported:

- A Starpoint SEC meter is run from a dedicated connector which matches pin for pin the connector on the back of the meter. Paylink provides facilities for reading updating and controlling all 32 possible counters within the meter unit.
- A number of mechanical / pulse meters can be connected to Paylink's standard outputs. There are controlled by the same interface as the SEC meter. Provision is made for continuing large updates over a power cycle, although the limitations of the way power fails can lead to the loss of a single pulse.

System Structure

A system that uses a Milan / Paylink unit comprises a user application that communicates via a DLL to the Milan peripheral handling code, either installed on a standard Paylink unit, or running as a part of the Driver program.

Either style of system requires a driver program to be running. This is usually a program, "Paylink", directly executing the peripheral handling code or communicating over a USB cable to a standard Paylink. For some Linux systems this driver program only communicates over a USB to a standard Paylink and is called AESCDriver.

USB Connection

USB Lead

The Standard Paylink unit is not designed to be added to and removed from a PC, it is designed to be permanently connected. Any disconnection and reconnect of the lead will at least cause exception processing, will usually cause a Standard Paylink unit to be reset and may cause the interface presented to the PC to change drastically.

The system should not be regarded as usable for 20 seconds after a Standard Paylink reset, or for 10 seconds after a USB driver program (re-)start.

USB Driver Program.

The supplied USB driver program Paylink (AESCDriver) has to be run, and should be regarded as a system service and unconditionally started at system boot.

It is possible to stop and start this program, but that always causes exception processing to be undertaken as above, and is not recommended. If it is not running and the Paylink unit is plugged in, then the Paylink units will continually turn their USB interface off and on in attempt to recover USB communications.

The system should not be regarded as usable for 20 seconds after a Paylink (re)boot, or for 10 seconds after a USB driver program (re-)start.

Paylink.exe runs under Windows where it also supports Paylink Lite 2, Paylink MDB Lite and USB peripherals made by CPI. On Linux the default driver is AESCDriver for use with a standard Paylink unit on **any** Linux system, and Paylink which is provided in a semi compiled form for Intel and Raspberry Pi based Linux systems.

The expected use of the driver program is that during initial program development, the driver is run and the program window is referred to in order to monitor and control the connection to the Milan / Paylink. When the system approaches a live configuration the driver program is run silently with a log file being produced for incident investigation. Finally, the launch of the driver is placed into the system start up files, where it should be regarded as a system service and unconditionally started at system boot.

Paylink Lite

Original (old) Paylink Lite

This is a small unit that runs very limited and old cctalk only firmware. It is still supplied by CPI to legacy customers. There is no configuration and only support for basic peripherals.

Paylink Lite V2 ccTalk

A ccTalk Paylink Lite unit is **completely** plug compatible with the cctalk connectivity on a standard Paylink. It has **exactly** the same USB connector, 6 pin cctalk connector and 12V connector, with exactly the same auto-resetting fuses on the 12V line and the 24V cctalk line.

The digital I/O is different from the standard Paylink, and features a 20 way connector which provides the following 5 groups of 4 pins:

- 4 digital inputs which are each the same as one of the 16 digital inputs on a standard Paylink
 - 4 pins connected to ground for use with these inputs
 - 4 digital outputs which are each the same as the 8 low power outputs on a standard Paylink; connecting the pin to ground when driven from Paylink.
 - 4 pins which are pre-wired for an LED, featuring a resistor connecting the pin to the 5V USB supply; so an LED requires no additional circuitry and can just be connected from here to an output pin – there is no equivalent of these pins on the Standard Paylink.
 - 4 pins connected to the 12V supply, for convenience when using higher power outputs – these are equivalent to the pins on a Standard Paylink.
-

Paylink MDB Lite

Rather than the limited 3 pin connector of a standard Paylink, the Paylink MDB Lite electronics are re-designed from the ground up to support MDB in an easy, cost efficient way.

The unit is fitted with the standard 6 way MDB connector, so that peripherals can be plugged directly into the unit without using any additional cabling.

A high power barrel connector is also fitted to the unit, to easily provide power to the MDB peripherals without any extra connections.

This connection is actually electrically isolated from the rest of the unit and is not necessary for the communications to function. In conjunction with the opto-isolation used on most MDB peripherals, this results in a very low noise system with no connection between the PC and the peripherals.

The 20 pin digital I/O connector on the MDB board is the same as the cctalk unit, except that there is no access to a 12V supply, so these pins is replaced by direct connections to the USB 5V supply, through an auto-resetting 500mA fuse.

Paylink Lite V2 RS232

Available only on special order for a system with no cctalk or MDB peripherals, an RS232 Paylink Lite V2 is available, with the same 20 way I/O connector and a 4 pin microfit connector for the serial connection.,

Paylink Lite Aux units

For configurations that require more than one connection, but where it is desired to still use the Lite system, auxiliary units are available which can be used in the following ways:

Lite Units in the system	cctalk	MDB	RS232
Base Lite V2 only	x		
Base MDB Lite only		x	
Base Lite V2 RS232 only			x
Base Lite V2 + Aux MDB	x	x	
Base Lite V2 + Aux RS232	x		x
Base MDB Lite + Aux RS232		x	x
Base Lite V2 + Aux RS232 & Aux MDB	x	x	x

(An Aux RS232 connection is handled by a specially programmed USB / RS232 converter, an Aux MDB is a specially programmed MDB Lite, with no I/O)

Paylink Lite is suitable if you use high power outputs, a lot of I/O, or the SEC meter.

User control of driver program

The Paylink system design expects the driver program to be launched during the system boot up process and then to run until system closedown.

The driver and the application (Paylink dll) communicate using a shared memory segment which is set up by the driver program once it is successfully communicating with the Paylink unit. The **OpenMHE()** call checks for the final "ready to go" flag set by the driver at the end of this process and returns 0 when this is found.

If the Paylink driver program exits and the application continues to run, then this shared memory segment remains initialised and a subsequent call of **OpenMHE()** may return 0, even though the Paylink system as a whole is not functional. Although in a normal system there is no reason for this to happen, you have to be aware of it if you decide your application itself will control the stop and start of the driver program. If you do this, then you should follow these rules:

If you intend your application to terminate the driver program, it should first call the **USBDriverExit()** function. This will enable a clean shutdown with any log file buffers written to disc. The application can monitor this close down by calling the **USBDriverStatus()** function and waiting for the returned value to change from **DRIVER_RESTART** to **USB_IDLE**.

Should your design call for your application to start the driver program, it should first check the return from an **OpenMHE()** call. If this returns zero, rather than the expected non-zero, then it should call **USBDriverExit()** to clean up the shared data segment before starting the driver program; as either another driver instance is currently running or a previous instance has not terminated properly. Under either circumstance the ensuing driver program startup will first mark the segment invalid (i.e. cause a non-zero return from an **OpenMHE()** call) before enforcing a 2 second delay (to allow any previous instance to exit) and then initialising the segment (causing a zero return from **OpenMHE()**).

If **OpenMHE()** returns the expected non-zero, then you can just start the driver program and wait for an **OpenMHE()** return value of zero.

As a final check before regarding the system as fully operational, the application can check for a status of **STANDARD_DRIVER** returned from **USBDriverStatus()**.

Troubleshooting

As detailed below in the configuration section, the Paylink driver program provides for a log file of limited size to be produced. Where the behaviour of Paylink is unexpected, this file will normally contain information that allows support personnel to establish precisely what has happened, to provide advice on how to stop it happening again.

It is therefore *strongly* advised that all operational systems set the driver up to produce such a log on writable permanent media, so that it can be sent to allows support personnel if necessary

Supported Peripherals

Paylink supports a wide range of peripherals. Most peripherals connect to the system via a serial communications cable plugged into a Paylink device, but some are connected directly via USB cable.

Note that hoppers and acceptors from multiple manufacturers support the same cctalk protocol, but bill recyclers are more variable. Each bill recycler below specifies the protocol with which it can be driven, a bill acceptor driven on one protocol will probably not be usable on a different one. Especially, we do not implement a multi-roll cctalk protocol.

Paylink Lite V2, which is available in three models with a single connection that is either cctalk, MDB or RS232, allows the connection of any supported peripherals that use the connection.

Some peripherals can only be directly connected through a USB socket; this can either be in addition to either Paylink device, or used with a µPaylink “Dongle”. This is only available for Windows and Intel or Raspberry Pi Linux systems.

The Standard Paylink unit has insufficient memory space to handle all the peripherals which are now supported. There are therefore 4 different firmware builds to cover the entire repertoire. The names on these builds are mostly historical accidents.

This table shows all the peripherals which are supported by the Paylink system, alongside the firmware build(s) that include them – note that USB connections are always available with all firmware builds:

	Genoa...	InnEbd...	Innov...	Mcd...	F5x...	USB
Coin Acceptors						
All cctalk coin acceptors	✓	✓	✓	✓	✓	✗
Coin Dispensers / Hoppers						
All standard cctalk Hoppers (see below)	✓	✓	✓	✓	✓	✗
CPI TFlex Coin Dispenser	✓	✗	✗	✗	✗	✓
CPI CX25 Coin Dispenser	✓	✗	✗	✗	✗	✓
Innovative SmartHopper in in CC2 (cctalk) mode	✗	✓	✓	✓	✓	✗
Coin Recyclers						
All MDB Coin Changers connected over MDB	✓	✓	✓	✓	✓	✗
Innovative SmartHopper with attached acceptor	✗	✓	✓	✓	✓	✗
CPI BCRxxx coin recycler	✗	✗	✗	✗	✗	✓
CPI CRxxx coin recycler	✗	✗	✗	✗	✗	✓
CPI CLS	✗	✗	✗	✗	✗	✓
CPI C2 MDB Coin Changer	✗	✗	✗	✗	✗	✓
CPI CF7000 MDB Coin Changer	✗	✗	✗	✗	✗	✓
Bill / Note Acceptors						
All cctalk Note acceptors	✓	✓	✓	✓	✓	✗
All ID003 Note Acceptors	✓	✓	✓	✓	✓	✗
All MDB Note acceptors	✓	✓	✓	✓	✓	✗
All EBDS Note Acceptors	✓	✓	✗	✗	✓	✗
All CCNet (CashCode, B2B) Note acceptors	✓	✗	✗	✓	✗	✗
Bill / Note Dispensers						
Fujitsu F53	✗	✗	✗	✓	✓	✗
Fujitsu F56	✗	✗	✗	✓	✓	✗
MFS series	✗	✗	✓	✗	✗	✗

Bill / Note Recyclers						
CPI / MEI EBDS SCR recycler, Normal, MNE and MDR	✓	✓	✗	✗	✓	✗
CPI / MEI BNR Recyclers (via manufacturers DLL)	✗	✗	✗	✗	✗	✓
All CCNet (CashCode, B2B) Note recyclers	✓	✗	✗	✓	✗	✗
JCM UBA, iPro & Vega recyclers on ID003	✓	✓	✓	✓	✓	✗
All MDB Note recyclers (on MDB)	✓	✓	✓	✓	✓	✗
ICT BR2300 Note recycler on cctalk	✗	✗	✓	✗	✗	✗
Single roll JCM Vega on cctalk	✗	✓	✓	✓	✗	✗
Innovative Single roll NV11 in cctalk mode	✗	✓	✓	✓	✗	✗
Merkur MD100	✗	✗	✓	✗	✗	✗
Innovative SmartPayout (NV200) in CC2 (cctalk) mode	✗	✓	✓	✓	✗	✗
Others						
Gen 2 ticket printer from FutureLogic	✓	✗	✓	✗	✗	✗
SEC Electronic Meter	✓	✗	✓	✗	✗	✗
Electromechanical Meters	✓	✗	✓	✗	✗	✗

Notes:

1. Where the description is for all peripherals of a type using a specified protocol, it is implied that the peripheral only uses the standard command set, as defined by the protocol "owner".
2. Cctalk hoppers are known to have two slightly different command interpretations, that used by CPI Serial Compact and Universal hoppers and that used by Azkoyen Hoppers. This has to be specified in the configuration file for completely correct operation.
It is possible that there are other hoppers which may have a different interpretation of the commands.
3. Paylink is *known* to operate correctly with many different coin and note acceptors on the MDB, cctalk and ID003 protocols. Apart from the MDB protocols, there is no guarantee that a given manufacturer's Bill / Note *recycler* will use protocol extensions that are known to Paylink.
4. If you have a requirement for a different mixture of peripherals, please contact us at Aardvark, it is not difficult to produce firmware with a new combination.

Coin / Note Acceptor Usage Details

Token Handling (Coin Ids) (1.11.x)

As tokens do not have a known value, they appear as coins with value zero. The only way for an application to detect tokens is to use the **CurrentUpdates()** function to detect activity, and then to check for increases in the count of the token(s) accepted(**Coin.Count**).

The index for the coin that holds the count for a particular token can be obtained by searching the coin array belonging to the acceptor and comparing the coin name (**Coin.CoinName**) with that of the token.

Dual Currency Handling (Coin Ids) (1.11.x)

If an acceptor is being used to accept coins or notes of more than one currency, the application can determine the currency of a specific coin or note by examining the name of the coin (**Coin.CoinName**) - usually the characters at the start.

Note: The exact values returned are dependent upon the acceptor manufacturers and hence cannot be given here.

ccTalk	This contains up to eight characters as returned by the Request Coin Id (184) command.
ID-003	This contains a representation of the three bytes as returned by the Get Currency Assignment (0x8A) command. The first three characters are the decimal value for country code, then a '/', then the base value as a decimal number, followed by a '^', then the count of extra zeros as a decimal number.
MDB	All MDB coins are the same currency. The coin name contains the Value as a decimal number, followed by a * followed by the (constant) Scaling as a decimal number
CCNet	This is set from the Get Bill Table (41H) command. The string is the 3 chars from the 3 byte "Country Code" followed by the decoded value as a decimal number.
EBDS	This is set from the reply to the Extended Note Specification message(0x02). The orientation character is always removed from the reply. EBDS acceptors can return a large number of identities (and Paylink can be configured to process them). Where Paylink can process all the identities the remaining 14 characters are used as the name, otherwise Paylink uses the 1st 10 characters of the name to merge the large number of identities into a smaller number.
MEIBNR	These are taken from the DenominationList, the Currency Code, Value and Variant are concatenated to give the string used.

Coin Routing.

Paylink provides facilities to partially automate the externally provided routing of coins to fill one or more coin dispensers and the cash box. This enables Paylink to accurately change the routing in the potentially very small delay between one coin and the next.

These facilities only apply to coins. The routing for notes is completely left to the application, as there are no time constraints.

There are 3 routing techniques:

- Route coins to a general cash box.
- Route specific coins to a specific cash box.
- Route specific coins to a dispenser until it is full then route it to a coin specific cash box.

There are 3 settings for each coin that are important:

- Coin.Path The path to the coin specific hopper.
- Coin.DefaultPath The path to the coin specific cash box.
- Coin.PathSwitchLevel When Coin.PathCount reaches Coin.PathSwitchLevel coins are routed to the coin cash box.

Route coins to a general cash box

- Set all coin paths to the desired route.

e.g. General Cash box on route 4.

- Path 4 for all coins
- DefaultPath 0 for all coins
- PathSwitchLevel 0 for all coins

Route specific coins to a specific cash box.

- Set Coin.Path for each coin that is routed to a specific cash box.
- The other 2 coin settings are zero.

e.g. General Cash box on route 4, coins 1 and 2 have separate cash boxes on routes 5 and 6.

- Path 5 for coin 1, 6 for coin 2 and 4 for all other coins
- DefaultPath 0 for all coins
- PathSwitchLevel 0 for all coins

Route coins to a hopper until it is full then route it to a coin cash box.

- Set Coin.Path to the hopper routing for each coin that is routed to a hopper
- Set Coin.DefaultPath to the cash box route for each coin that is routed to a hopper. **This must be non-zero**
- Set Coin.PathSwitchLevel to the Coin.PathCount value at which the hopper will become full. **This must be non-zero**

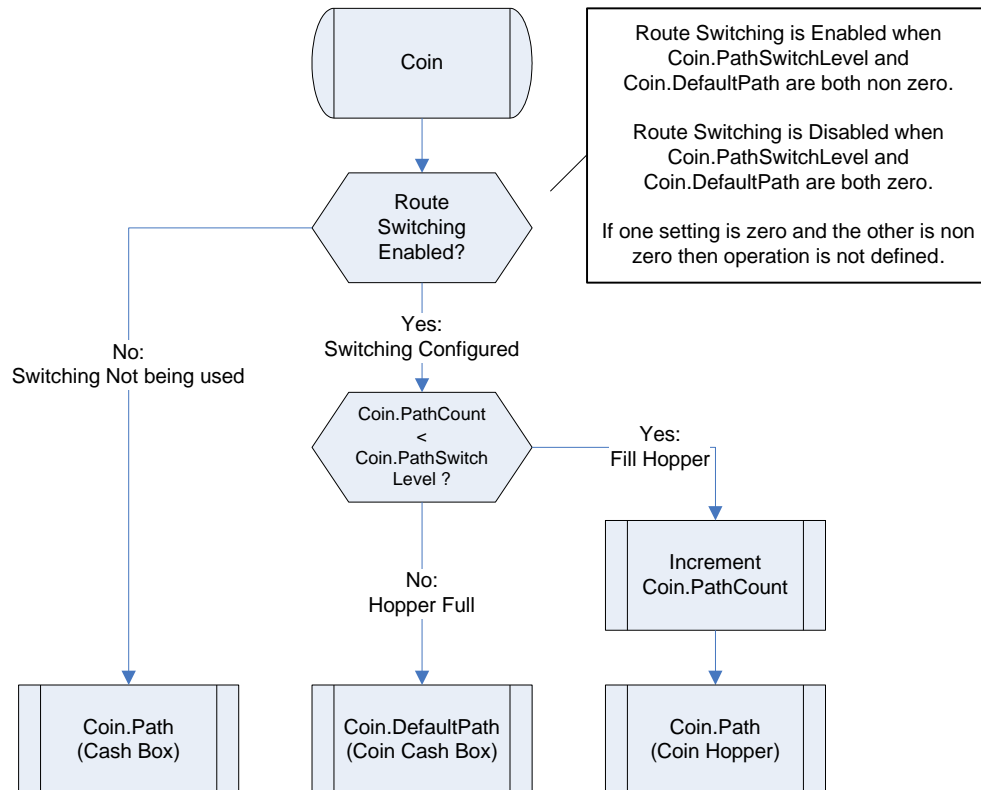
e.g. General Cash box on route 4, coin 1 goes to a hopper on route 1 and a cash box on route 2. Coin.PathCount is 100 and there is space for 300 more coins in the hopper.

- Path 1 for coin 1, 4 for all other coins
- DefaultPath 2 for coin 1, 0 for all other coins
- PathSwitchLevel 400 for coin 1, 0 for all other coins

When coins are routed to the dispenser (via the Coin.Path route) the variable Coin.PathCount is incremented. When PathCount reaches PathSwitchLevel, further coins are routed to the coin cash box. As the dispenser pays out coins the PathSwitchLevel should be increased by the corresponding amount. Further coins will then be routed to the dispenser again until the new switch level is reached.

As this system relies on PathCount and PathSwitchLevel accurately tracking the contents of the hopper, Paylink saves these numbers in non-volatile storage - they therefore reflect counts for the life of the unit.

Paylink Routing - Flow Diagram



Notes:

- **Setting route 0 should be avoided as it does not exist on an SR5 coin acceptor.**
- **The settings for PathSwitchLevel and PathCount are restored automatically by Paylink after a reset.**

Control of Motorised Acceptors

ccTalk bulk coin acceptor (1.11.3)

Paylink will automatically send the required command to operate the motor on a bulk coin acceptor, but the application may want to trigger the special “reject clearance mode”. This is requested by inhibiting all the coins for the acceptor *and* inhibiting the acceptor itself. (To *just* stop accepting all coins, it is only necessary to inhibit the acceptor)

BCR / CR10x coin recyclers (1.11.5)

There are a number of features of these devices that require special handling by Paylink. Specifically Paylink will:

- Issue a carousel clear to a CR10x 2 seconds after the unit is disabled, if a Payout has not been requested.
- Respond to a BCR fault reports with a subsystem clear command
- Report the unit is busy while any of the carousel, singulator, payout belt etc. are active

MDB changer / BCR / CR10x / CLS recycler / SmartHopper support.

If a coin changer / recycler is used, it will appear as an acceptor in very much the same way as any other acceptor. The coins that are routed for recycling can be distinguished as having a non-zero Routed Path, although, obviously, any changes made to the routing by the application will be ignored.

For Payout, the application will (either explicitly or implicitly) specify exactly which coins are to be used for the payout. Where higher value coins are not available, the standard Paylink algorithm will work down the values of the coins as usual.

MDB Payout

For MDB payout, the situation is slightly complicated. The MDB changer protocol supports two different payout mechanisms, a basic one that is always present and an extended level 3 one, which is supported on any current changer. The basic system provides control over the individual payout tubes, but has no feedback as to whether the payout works. The extended one provides feedback as to the success of the payout, but does not allow any control over which tubes the payout is from.

The solution adopted is to always provide one dispenser for each tube, which is run using the basic mechanism and for the extended mechanism an additional dispenser is provided which is run using the extended mechanism. Alongside the extended mechanism dispenser, the individual level 2 tubes are pre-set as inhibited.

To perform a “normal” payout, you just issue a **PayOut()** request and call **PayStatus()** and **CurrentPaid()** to monitor the results. (If you use level 2 dispensers, **CurrentPaid()** will update almost instantaneously rather than at the end and will always show that all coins requested have been paid, regardless of the actual outcome.)

If you use thy normal level 3 changer, **CurrentPaid()** will update during the process, and you may get a PAY_EMPTY status from **PayStatus()**, with **CurrentPaid()** then reflecting the actual payout achieved.

The current levels of MDB tubes, *as reported by the coin-changer*, are returned in the field **CoinCount**. In addition, the field **CoinCountStatus** will contain the value DISPENSER_ACCURATE for a normal tube, and DISPENSER_ACCURATE_FULL if the changer is reporting the tube as full. Note that the levels reported by the changer do not necessarily update in a “sensible” fashion after a payout.

Should you wish to perform an operation on a specific tube (e.g. emptying it), you should inhibit the extended mechanism dispenser and enable the specific tube you wish to control.

As the manufacturer is already shown in the acceptor detail block for the changer, the extended mechanism dispenser has a **Unit** field with the constant value of **DP_MDB_TYPE_3_PAYOUT** while the individual tubes have **Unit** fields with the constant value of **DP_MDB_LEVEL_2_TUBE**.

MDB tube level monitoring.

Monitoring:

The main method for determining tube levels is via the Tube Status (0x02) MDB command.

This is issued during startup and then every 25 seconds. The response to this is copied directly into the tube coin level, and one of the DISPENSER_ACCURATE or DISPENSER_ACCURATE_FULL level statuses set.

Coin Insertion:

When a coin insertion (MDB event code 0x40) is reported as going to a tube, the changer also includes an updated value for the tube level. If this is non-zero then this is used to overwrite the coin level for the tube. . (Note that after a delay this will then be *replaced* by the value from a Tube Status command)

When a coin insertion is reported as going to the cashbox for a coin that has an associated tube, Paylink immediately issues a Tube Status (0x02) MDB command to obtain an accurate value for the levels.

Manual Dispense:

When a manual dispense (event code 0x80) is reported then the reported tube level copied directly into the tube coin level. . (Note that after a delay this will then be *replaced* by the value from a Tube Status command if that is different)

Payout:

While a payout is in progress, no updates are made to the coin level. As soon as the payout completes, Paylink immediately issues a Tube Status (0x02) MDB command to obtain the changer's opinion of the new levels.

Read out of Acceptor Details (1.11.x)

Different protocols / manufacturers provide different details on acceptors. The **Acceptor.Description** field is generated from the information provided as follows:

ccTalk	The replies to: <ul style="list-style-type: none"> Request Currency Revision / Issue (145 / 96+243), Request Currency Specification ID / Code (91 / 96+244), Request Software Revision (241) & Request Product Code (244) commands, separated by '~' characters. Each individual field is omitted if there is no response to the command, although the '~' character is still inserted.
ID-003	The entire reply to the "Get Version Request" (0x88) command
MDB	From the Status and Extended Identification Commands <ul style="list-style-type: none"> Country Currency Code (4 BCD characters) Decimal Places (1 Character) Manufacturer (3 Characters) Model Number (12 Characters) Software Version (4 characters) separated by '~' characters.
CCNet	This is the 15 character "Part Number" from the "Identification" (37H) command.
EBDS	The replies to: <ul style="list-style-type: none"> Query Type (0x04) Project Number from Query Application Part Number (0x07) Firmware Version Number from Query Application Part Number (0x07) separated by '~' characters.
MEIBNR	Is taken from the first component returned by GetIdentification. It is the Description followed by the Version major and minor

The **Acceptor.SerialNumber** field is generated as follows:

ccTalk	The binary reply to the ID Serial No (242) command.
ID-003	The "standard" ID-003 protocol does not allow for a serial number. A non-standard 0x8F query is issued and any response will be stored here.
MDB	Bytes Z4-Z15 from the Extended Identification Command, converted from decimal characters to a number.
CCNet	The "Chassis Serial Number" from the Module Identification Request (53H0) command, converted from decimal characters to a number.
EBDS	The reply to Query Serial Number (0x07). Unfortunately this doesn't fit into a 32 bit number, so digits 5 & 6 are ignored.
MEIBNR	Is taken from the first component returned by GetIdentification It is the numeric value of the 1st 6 characters of the name field.

Coin / Note Dispenser Usage Details

Dispenser Power Fail support.

Some dispensers, especially hoppers produced by MCL and some bill recyclers, are guaranteed to correctly count coins even if power is removed during a payout sequence. This facility is explicitly supported in the Paylink software. The `Count` field in the interface for these hoppers is set during Paylink start-up initialisation to correspond to the “total coins paid since manufacture” value (or its closest equivalent) retrieved from the hopper, and is then updated as payouts occur. This field allows for the correct counting of coins over a power failure.

At the end of every payout sequence, the Paylink stores, internally, the `Count` for each hopper. At initialisation as well as reporting the retrieved count, it is also compared with the saved value. This enables the `CurrentPaid()` function to continue to report the correct value, and also generates an `IMHEI_COIN_DISPENSER_UPDATE` *Event* (see below) to register this update.

Detailed Device Support.

Abandoning a payout in progress (1.11.3)

As well as preventing a payout operation from starting, the `Inhibit` field in a dispenser is also used during an actual payout. If the application set a dispenser inhibit while a payout it is in progress, Paylink will attempt to abandon the payout in progress on that device.

Note that the overall payout will still continue on all the other dispensers that are not inhibited. To cancel an entire payout the application should inhibit all dispensers.

Control of unwanted bill payout (1.11.3)

Under failure conditions a number of bill handling systems can enter a state where bills are not accessible to the end user, but cannot be returned to a cash / reject location. When Paylink detects these circumstances, it will pause its operation, queue a `IMHEI_NOTE_DISPENSER_PENDING` event with the number of bills as the `RawEvent` field and automatically set an inhibit on the relevant dispensers.

The bills can be delivered by clearing the inhibit for *all* the dispensers that form part of the unit.

Combi Hopper Support.

This single unit is no longer actively marketed, but dispensed two different coin values. It is therefore handled in a similar way to the MDB system. There is a primary dispenser, which is set up as a normal unit with a `Unit` field of `DP_MCL_SCH3A`, and a `Value` field with the lower coin value in it. The `Count` in this dispenser is the count of the lower value coins dispensed. In addition, another dispenser is set up, with a matching `Address` field, a `Unit` field of `DP_CC_GHOST_HOPPER`, the `Value` of the higher coin and the `Count` of the higher value coins dispensed.

Note that, due to limitations of the unit, during a payout operation the `Count` of the main dispenser *only* is updated, as though all coins dispensed were of this value. At the end of the sequence, while `LastPayStatus()` is still returning `PAY_ONGOING`, the accurate count of both coins is retrieved and the two separate `Count` fields updates. The result of this is that, as the operation finishes, the `Count` for the lower value dispenser decrements.

Read out of Dispenser Details (1.11.x)

Different protocols / manufacturers provide different details on acceptors. For almost all recyclers the details are copied from the acceptor field.

The Description (**Dispenser.Description**) field is generated as follows:

ccTalk	The replies to: <ul style="list-style-type: none"> Request Software Revision (241) & Request Product Code (244) commands, separated by '~' characters. Each individual field is truncated to 15 characters, and is omitted if there is no response to the command, although the '~' character is still inserted.
F56 / F53	The 12 character firmware revision, followed a '~' followed by the 32 character device information.
MEIBNR	The physicalCashUnit name.

The **Dispenser.SerialNumber** field is generated as follows:

ccTalk	The binary reply to the ID Serial No (242) command.
F56	Not Available

Complex Dispenser (Recycler) Operations (1.12.3)

Introduction

The original Paylink model was based around the concept of independent acceptors and dispensers, with the main specification for a dispenser being the MCL / cctalk hopper.

The advent of bill / note recyclers meant that the Paylink model has had to be enhanced to include these. The approach adopted is based around the idea that bill / note recycler (and the coin changer MDB device) is a combination in a single unit of an acceptor and number of dispensers.

Outside of their core function, Complex Dispensers can have many more operational states, rather than just running / errored. In addition, whilst the error reporting system is the same as for simpler devices, there can be more details than the 12 or so pre-defined Paylink errors. The approach taken is that an appropriate simple error is reported, and the RawEvent byte details the exact error reported by the device. This section describes the operational characteristics of the devices, details on the reporting of events / faults are detailed in the appropriate subsection of the [Events \(Faults / Auditing\)](#) Section.

Security

The connections used by Paylink fall into four categories, cctalk, RS232, MDB and USB. These communications systems tend to match up with different markets:

cctalk is a very versatile system with a connection that is relatively vulnerable to interference.

Paylink has, as a general philosophy, the idea that it will connect to anything. Using the cctalk DES encryption facility in general for a bill and a coin acceptor do not therefore make sense. A DES bill recycler however makes sense, as the locked down aspect here is in the peripheral - Paylink can connect to any recycler, but the recycler will only communicate with Paylink.

Given the vulnerability to high value fraud of a cctalk recycler, Paylink therefore insists that any cctalk recyclers that can support DES encryption must be used with DES encryption turned on.

RS232 is used with a number of protocols to connect expensive bill acceptors / recyclers for applications that are typically in expensive, secure enclosures. Any interference with this connection will generally tend to be visible to the Paylink application.

MDB is used in very cheap systems, typically vending machines. The connection is vulnerable to interference, but the amounts of money involved tend to be low.

USB is used in many systems, but is relatively secure as intercepting operational communications is difficult to achieve.

DES Key Exchange

Before using a cctalk DES based device, Paylink has to acquire and store the random DES key provided by the device. Paylink automatically checks at device discovery whether it has a correct key, and if it hasn't Paylink goes into key exchange mode.

A Paylink device in key exchange mode flashes the green LED at twice the normal frequency, and a device waiting for a key exchange is visible to the application with the ACCEPTOR_NO_KEY bit set in the AcceptorBlock.Status field.

The mechanism for triggering a key exchange is unique to each manufacturer / device.

Component Identity

The general approach to identifying recycler devices is that the acceptor part usually contains the overall description of the unit, and the dispensers are identified by an address (which is often a sequence number from 1 upwards that is an intrinsic part of the protocol) together with the value that they dispense. If the unit only has one dispenser, the address will be 1.

Where necessary a dispenser can be tied to acceptor by type and by having the same serial number. The `DispenserBlock.m_UnitAddress` field(s) will contain the above address that identifies the dispenser

On multi-drop system, such as cctalk and MDB, the `UnitAddress` will contain the address of the parent acceptor OR'ed with this dispenser address number so that multiple units can be distinguished.

Routing

Dispenser Destination Initialisation

Paylink, during its initialisation of any recycling unit, always determines the value of the coin / bill in the dispenser(s) and which coins / bills are routed into which dispensers. The address(es) of the dispensers are stored into the `AcceptorCoin.Path` field(s). All the other coins which route directly into the cashbox will **always** have `AcceptorCoin.Path` fields of zero.

When a coin / bill is accepted and routed into a dispenser this fact is **always** identified by Paylink and the `AcceptorCoin.PathCount` is always accurately incremented to show this.

After acceptance, Paylink then updates the `DispenserBlock.CoinCount` field by actually querying the unit. Depending upon the actual unit this will be either accurate or an approximation. With a bill recycler the result is usually an accurate figure, with an MDB changer the result is often approximate.

The value returned will however *always* be that reported by the device, any systematic corrections will have to be handled by the application.

Routing Control.

For all coin recycler units the routing is fixed and it is not possible for Paylink, and hence the application, to change this.

For bill recyclers units, the routing can in general be changed by Paylink. The application notifies Paylink of the desired routing by changing the `AcceptorCoin.Path` fields of the incoming Coin (Bill) array item to contain the associated dispenser address, as described in Component Identity above.

The precise actions that occur when an `AcceptorCoin.Path` field is changed:

- If the `AcceptorCoin.Path` field for a currently recycled bill is changed to zero, Paylink sets the unit to stop diverting bills into the recycler. If there are no bills stored, then the `Dispenser` value will go to 999999999, (this is irrelevant to payouts, as the dispenser will return "empty" if it is attempted to be used), if any bills are currently stored they *may* remain available to be paid out (depending upon the device capabilities.)
- If the `AcceptorCoin.Path` field for a currently non recycled bill is changed from zero to a non-zero value, Paylink will set the unit to start diverting these bills into a recycler. Where possible the recycler used to contain the bills will be that identified by the value in the `AcceptorCoin.Path` field - but this is device specific. This may cause different value bills already in that recycler to be dumped to the cashbox.

- If the `AcceptorCoin.Path` field for a currently recycled bill is changed to different non zero value, Paylink will attempt to set the unit to recycle the bills to a different recycle unit. This may cause different value bills already in that recycler to be dumped to the cashbox. If bills are currently stored on the previous recycler they *may* remain available to be paid out (depending upon the device capabilities.)
- If more bills have a non-zero `AcceptorCoin.Path` value (i.e. are set to recycle) than the unit has available recyclers, or if the values do not map onto available units, then Paylink does not update the unit and (silently) waits for the application to reduce / re-arrange the bills that are recycled. There will be a comment to this effect output to the Paylink log.
- The application should not set two or more separate bills to contain the same value in the `AcceptorCoin.Path` field, if this is done, then this situation is undefined

Note that as the specification is “where to send the bill” there is no simple method to have two dispensers regarded as receiving the same value bill - on some recyclers this *might* be achieved by changing the `AcceptorCoin.Path` to a different dispenser, which *can* leave the previously set dispenser receiving the bills.

The options for automatic re-routing using `DefaultPath` and `PathSwitchLevel` are only available with coin acceptors. *For note recyclers, these fields are **never** used.*

Bill Recycler Emptying

The high value represented by the bills in recyclers means that the dispensing of bills requires the interaction of the recipient. The high value of the bills stored in the recycler also means that users are liable to want to empty them at the end of the day.

These two factors mean that bill recycler manufacturers implement a “dump to cash box” facility so that the bills can easily retrieved.

Full Dump

A full dump is where the recycler takes every bill from a dispenser into the cash box until the dispenser registers as empty.

Triggering this is implemented on Paylink by the user setting a `DispenserBlock.Status` value of `DISPENSER_CASHBOX_DUMP`.

On recyclers that maintain guaranteed accurate counts of bill, the application can monitor the dump process by observing the `DispenserBlock.CoinCount` going to zero.

On both these and other recyclers, the application can check for the `DISPENSER_CASHBOX_DUMP` value being replaced by another status. Where the dump process completes normally, the status will take value of `DISPENSER_DUMP_FINISHED`.

Partial Dump

As well as the above facility to cycle every bill into the cashbox, many recyclers provide the ability to perform a partial cashbox dump processes, which can be used to leave a “float” of bills in the recycler.

Triggering this is implemented on Paylink by the user setting the count of bills that are to be dumped in the new `DispenserBlock.NotesToDump` field and a value of `DISPENSER_PARTIAL_DUMP` in the `DispenserBlock.Status` field.

The application can usually monitor the dump process by observing the `DispenserBlock.CoinCount` field reducing by the requested amount.

The application can check for the `DISPENSER_PARTIAL_DUMP` value being replaced by another status. Where the dump process completes normally, the status will take value of `DISPENSER_DUMP_FINISHED`.

Payout Progress

labelling Cancellling Payout

Bill recyclers / dispensers in general hold the bills awaiting collection by the user, and Paylink does not regard the Payout as complete until the bill has actually been taken. If the application program decided that the bill has been forgotten, it can abandon the payout by setting the inhibit flag on the dispenser. This will cause Paylink to request that the recycler abandons the payout and returns the bill to the cash box.

Note that following the abandonment of the bill payout Paylink will automatically proceed to attempt payout in coins or other bills, so in the usual case all the dispensers should be disabled at the same time.

labelling Notification of progress

While a bill recycler / dispenser is holding a bill awaiting collection by the user, Paylink does not regard the Payout as complete. The fact that the bill is available to be taken is however possibly of significance to the application, and therefore Paylink will update the `DispenserBlock.Count` and `DispenserBlock.CoinCount` fields for the relevant dispenser as soon as the bill is accessible. They will not then change when it is taken.

Power Fail

labelling Temporary power interruption

Should the power / communications to the recycler fail during a payout while Paylink continues to run, Paylink will initially just wait for the communications to restart, and will then continue as though there has been no interruptions.

Where an interruption lasts “a long time” then Paylink will abandon the payout attempt. This will result in a dispenser / payout status of `PAY_US`. If at the time the payout is abandoned, Paylink is aware of a bill awaiting collection by the user, it will be regarded as having been paid out. It will not therefore be substituted by coins and can result in a normal payout completion status.

After the timeout, if / when normal communication with the recycler is resumed, Paylink will check the current status of the unit.

- A bill that was paid and collected during the interruption will cause the `DispenserBlock.Count` field to be incremented by the appropriate amount and a `IMHEI_NOTE_DISPENSER_UPDATE` event entered in the `NextEvent()` queue.
- A bill that has been sent from a dispenser to the cash box (as a part of the start-up recovery process) will merely result in the `DispenserBlock.CoinCount` being updated.
- A bill that was awaiting collection, and has still not been taken, will cause the dispenser status to change to `PAYOUT_ONGOING` until it is eventually collected. This will have no effect on the `DispenserBlock.Count` field or Payout system.
- A bill that was awaiting collection but is **now** known to have been automatically recycled to the cash box, will cause the `DispenserBlock.Count` field to be decremented (to “undo” the payout) and a `IMHEI_NOTE_DISPENSER_UPDATE` event entered in the `NextEvent()` queue.

labelling Full power Failure

Should the power to PC and recycler fail during a payout the application may wish to reconstruct the partial results of the last payout attempt. To facilitate this, Paylink will attempt to handle the interrupted payout according to the above rules

To do this requires that the `DispenserBlock.Count` field be maintained over power cycles - thus applications that so desire can record the `Count` fields before a payout is started, and then react accordingly if on startup they discover that a payout was in progress.

With coin hoppers, the speed of the payout means that the only place where an accurate record of interrupted payouts can possibly be obtained is in the hopper itself. Note dispensers typically do not provide such facilities, and Paylink therefore maintains the record itself.

The net result is that lifetime totals are maintained and reported in the `Count` fields, which are retrieved and updated by Paylink depending on the data read from the device during startup.

If this startup processing causes Paylink to suspected an uncompleted payout actually completed, an `IMHEI_COIN_DISPENSER_UPDATE` event is entered in the `NextEvent()` queue.

Unpaid Bills

Some devices (e.g. the B2B-300 bill recycler and F56 bill dispenser) have a delivery stage where bills are accumulated for eventual payout.

Following a power failure, it can happen that bills are in this output stage and are inaccessible to the user. The only thing that Paylink can do at this point is to complete the delivery of these bills, but as there is a potentially long time since the application requested the payout that accumulated these bills, it is inappropriate to just deliver them at power up.

(Some F56 / F53 models also have a problem that following a power failure during a payout there can be an *unknown* number of bills awaiting delivery.)

In these cases, for each dispenser device that is believed to have notes ready for delivery, Paylink marks the device inhibited, and generates an `IMHEI_NOTE_DISPENSER_PENDING` event with the number of bills as the `RawEvent` field. If the number of bills is unknown then 99 is used.

To complete the delivery process, the application should clear the inhibit setting on **all** the relevant dispensers.

Device Specific Functionality

The above description is the ideal that Paylink strives to achieve. The actual functionality provided by specific devices can however interfere with this, so all supported models of note / bill recycler are itemised here:

MEI BNR

If a bill is set to not be recycled, any bills of that value already in the recycler(s) will be dumped to the cashbox.

The dispenser address of recycler units are those intrinsic to the BNR hardware and correspond to the "physical unit index" - typically these values are 3 to 6.

This bill recycler is capable of directing the same bill to multiple dispensers, using the method described above.

Paylink fully implements the partial dump described above, but a restriction in the operation of the BNR means that the BNR will reset if a dispenser is partially emptied to a level that is different to that used last time. (The BNR dump command dumps notes to a preset level - changing this preset level requires a reset.)

This device supports a cash loader, and also requires resets during normal operation. During both of these operations the device is not available for use. Paylink reports to the application the fact that the device is unavailable by marking the acceptor device as busy (ACCEPTOR_BUSY).

When the user of the BNR device accesses the loader or the cashbox, the device goes out of service. Paylink notifies the application of this by queuing an IMHEI_NOTE_STACKER_PROBLEM event to the application and a subsequent IMHEI_NOTE_STACKER_FIXED when the intervention is complete.

SCR Advance (EBDS)

The bill routing to recycler units are reported to the Paylink during startup and so the `AcceptorCoin.Path` value is initially set up correctly. When the `AcceptorCoin.Path` value is changed by the application, the SCR cannot be told which recycler to use, only that the bill is to be recycled. This means that when you write to these fields the values are essentially a zero / non-zero flag.

If only one bill is set to recycle, it will be automatically routed to both recyclers, with the unit deciding dynamically which one to use for a payout request. There is no way of overriding this behaviour.

All dump facilities work as expected.

The SCR has two special features that are unique to the device, both relating to the management of bills held on the recycle drum.

It is able to detect that a note held on a drum is not recognisable as a valid note. This is reported to Paylink, which notifies the application by queuing an IMHEI_NOTE_DISPENSER_UNRECOGNISED event, and then proceeds to automatically dump the bill to the cashbox. This sequence is repeated for each invalid bill.

During startup it is also capable of detecting that there are fewer bills stored on a drum than it was expecting. Paylink notifies this event to the application by queuing an IMHEI_NOTE_DISPENSER_MISREAD event to the application, which identifies the drum in question and includes the count of bills that are missing.

SCR Advance Mixed Denomination Recycler (1.12.12)

This is the MDR variant of the standard SCR which allows Paylink to have complete control of the 2 recycle drum units.

Notes are initially automatically accepted onto drum 1, but can then be moved to drum 2 (and back again) under Paylink's control.

Paylink always operates in terms of dispensers, each of which normally corresponds to a physical device (or part thereof) but each of which has to contain a *single* bill denomination. As in the MDR the actual recycler rolls (can) contain multiple bill denominations, each Paylink dispenser corresponds not to a drum but to a single bill denomination. If the recycler can handle 7 different denominations then there will be 7 of these "virtual" dispensers.

All the details of moving notes between drums to achieve payouts are handled by Paylink. The API presents a "virtual" dispenser unit for each denomination handled by the recycler, there are no Paylink control blocks that map onto the actual physical drums inside the recycler.

The application can specify which drum it would like notes to be initially held on for storage, by setting either 1 or 2 as the Routed Path for the denomination. (Path values of zero are sent straight to the cashbox.) At present the *device itself* limits recycling to a maximum of 4 different denominations.

When Paylink receives a payout (or dump) request, it checks the two drums to find a bill involved in the payout that is closest to the "top" and then pays out (dumps) that one bill. It then restarts this process for the remaining bills. This repeats until the payout (or dump) is complete.

If necessary in order to payout this bill it will move any bills "above" it to the other drum (or dump them to the cashbox if the other drum is full). Importantly they are *not* returned afterwards to their acceptance drum.

The choice of denominations that are present in the payout is the standard Paylink one of working down in denomination value and planning to pay as much as possible of each value before proceeding to lower values. (This default can be overridden by use of the Exact Pay facility, but it is not expected to be.)

Importantly, Paylink does *not* examine the payout as a whole with regard to which bills are held on the rolls. Specifically, it does not re-assign the denominations required based on the current contents of the rolls. Although there may be a theoretical benefit to this, in practice the complexity outweighs any advantage.

The MDR recycler also has the same special situations at startup as described above for the SCR.

JCM Vega (cctalk DES) & Innovative NV11 Recycler (DES)

This is a standard, single bill recycler, with no special features.

These recyclers can retrieve a bill waiting for collection, and send it to the cash box.

If the dispenser is inhibited during a payout then, as well as preventing further payouts, Paylink will actually retrieve the bill waiting for collection. As this occurs after the bill has been accounted for, both the `DispenserBlock.Count` field and the `CurrentPaid()` return value will decrement.

(Note: although the JCM Vega in cctalk mode is limited to a single bill recycler, the dual recycler model *is* supported in ID-003 mode.)

ICT BR2300 (Not available with DES)

This is a completely standard, single bill recycler, with no special features. It isn't available with DES encryption, so runs in the much less secure BNV encryption mode.

MDB Note recycler

MDB note acceptors with recycler capability are automatically detected and supported.

The MDB specification is much more limited than other protocols and so the MDB Acceptor does not allow for:

Control of bill routing. The recycler reports the set-up of bill routing, allowing for the acceptor routing information to be initialised, but altering this has no effect.

Bill Dump.

It is not possible to initiate a recycler to cash box note dump so the DUMP and PARTIAL_DUMP operations have no effect.

Innovative NV200 Recycler / SmartPayout (DES)

This recycler by design stores all bills into a single storage space. To allow for control over the payout operations, Paylink treats each denomination as stored into a separate “dispenser”; so each denomination is set up as routing into a matching dispenser.

To stop storing a particular denomination, the routing can be zeroed - and to empty all bills of a particular denomination into the cash box, the corresponding dispenser can be dumped.

Innovative SmartHopper coin recycler

This recycler by design stores all coins into a single storage space. To allow for control over the payout operations, Paylink treats each denomination as stored into a separate “dispenser” of type DP_SHOPPER.

In addition there is a special Dispenser DP_SHOPPER_TOTAL for overall control.

When dumping from smart hopper, you can either do a controlled dump, emptying all or some coins of a particular denomination into the cash box, by using `DispenserBlock.NotesToDump` and setting `DispenserBlock.State = DISPENSER_PARTIAL_DUMP` on the individual DP_SHOPPER. Note that the `DISPENSER_CASHBOX_DUMP` command does not work on the individual DP_SHOPPER.

(In order to allow multiple denominations to dump simultaneously, Paylink accumulates dump requests for a period of ½ seconds before issuing the corresponding command.)

To empty all the coin in the smart hopper into the cash box you should use `DispenserBlock.State = DISPENSER_CASHBOX_DUMP` on the single DP_SHOPPER_TOTAL dispenser.

When used with an integrated acceptor or coin feeder, each denomination is set up as routing into the matching dispenser. To stop storing a particular denomination, the routing on that denomination can be zeroed in the `AcceptorBlock`.

If you use the integrated acceptor model, it may be necessary to periodically dump the unwanted denomination(s).

JCM UBA & iPro Recycler

The UBA and iPro recyclers are functionally identical.

These recyclers can retrieve a bill waiting for collection, and send it to the cash box.

If the dispenser is inhibited during a payout then, as well as preventing further payouts, Paylink will actually retrieve the bill waiting for collection. As this occurs after the bill has been accounted for, both the `DispenserBlock.Count` field and the `CurrentPaid()` return value will decrement.

This unit can have bills “manually” loaded into the storage stackers. When this occurs, the unit doesn’t know about the event and so does not update its internal counts, these are only updated when bills that have been accepted are stacked.

Similarly, if bills that have been stacked automatically are manually taken, the counts are not reduced.

The counts returned by Paylink are those from the unit, and so under these circumstances will be incorrect - it is up to the application to compensate for those bills it knows have been manually inserted.

As a special facility *only for the iPro / UBA* the coin count in the actual device can be updated from the API. If the new value is written into `DispenserBlock.CoinCount` and the `DispenserBlock.Status` is set to `DISPENSER_UPDATE_COUNT` then Paylink will attempt to update the count in the device and will set the `DispenserBlock.Status` to `DISPENSER_COUNT_UPDATED` if it is successful.

When the counter of the number of bills in a recycle stacker reaches zero Paylink will still attempt to pay bills - if this succeeds the UBA counter will remain at zero - it will not go negative.

Similarly, if the recycle stacker runs out of bills when the counter is non-zero, the will zeroize the counter.

Note that the recycler itself uses the routing / recycler specification during payout. All notes paid out undergo the normal input validation process and any that fail this validation are stack into the cashbox without comment.

F56 / F53 Bill Dispenser

The F56 device comes with a number of different options. Although the F53 is a different device number, Paylink just regards it as another option of an F56. All descriptions are therefore of an F56.

The F56 has a number of unique characteristics:

- The F56 only reports cassettes that are present; the existence of a location for a cassette is not discoverable. Paylink therefore only reports the status of cassette locations in which it has seen a cassette.
- A cassette can have a pattern of magnets set into it to indicate the type of bills with which it is loaded. The F56 configuration can include bill descriptions corresponding to these magnet patterns, which can specify value, bill length and bill thickness. If a newly discovered cassette matches such a pattern specification, then the bill value and sizes are set from the specification.
- If no magnet specification is given, or if there is no match, then the sizes default to a generic accept all size and the value is set as 999999999. This can be overridden to its correct value using the standard Paylink facilities.
- A pool area / note delivery option is possible, with delivery to the front or to the rear. Part of the configuration specification of an F56 has to include whether or not a delivery option is fitted.
- The F56 records in non-volatile memory the number of bills delivered from a payout position. This value is reported to the application in the `DispenserBlock.Count` field.
- Some F56 models allow for the recovery of a failed dispense operation - on others this information is not available. Where this information is not available and the unit has a final dispense stage, then the notes are left in the pool area and **Unpaid Bill** processing performed.
- An F56 can be fitted with a shutter at the bill delivery stage. Paylink will automatically send a close shutter command when bills have been taken from the delivery stage by the user.
- An F56 can reject bills as they are being paid out. Although it does not “fit” the API as specified, the handler actually returns the cumulative total of the in the **DispenserBlock.CoinCount** field.

F56 / F53 Jams

The F56 is complex mechanically, so a jam situation can be reported for a number of specific reasons. To allow the application to handle this the F56 handler will generate specific event (via the `NextEvent()` system) to notify the application of these. Details on this can be found in the [F53/F56 Fault Processing](#) subsection.

Cashcode B2B-300

The Cashcode B2B-300 accumulates bills to be dispensed in separate unit before presenting them to the user. When bills are “found” in the dispenser during startup, the only thing the unit *can* do is to dispense them.

When Paylink discovers this situation, during startup, or following a “long” power fail, it will undertake **Unpaid Bill** processing as above.

Cashcode B2B-60

The Cashcode B2B-60 operates by paying bills one at time for retrieval through the acceptor. To allow for full control in the event of a power failure / connection problem, Paylink runs the acceptor so that each note is a separate transaction. This interacts badly with coins if you set the simultaneous hoppers flag.

A B2B60 specific option is to actually “throw” the bills from the acceptor, rather than letting them be removed by the user. This processing can be requested by an option in the Paylink configuration file.

Merkur 100

This recycler automatically restarts a payout on power up, unless a software reset is issued before the acceptor reaches the point at which the delivery is under way.

During the startup process, Paylink issues such a reset, so if the two units power up approximately at the same time, no spurious bill is paid. If this succeeds, then any bills “in progress” will be returned to the stacker.

This recycler is capable of directing the same bill to multiple dispensers, using the method described above.

Extended Escrow (1.12.6)

Introduction

The original design of the Paylink API pre-dates the arrival of note recyclers; at the time that Paylink was produced coin escrow was handled by “external” equipment controlled by a simple output driving a solenoid.

The Escrow facility implemented in all Note acceptors provides a way of double checking that a note is acceptable before it is accepted, but does not provide a proper generalised escrow facility as:

- There is a significant delay between accepting the escrow note and knowing that it will remain inaccessible to the user
- It only provides escrow for a single note

Paylink now provides an ExtendedEscrow API which, when used in conjunction with a suitable note recycler, meets the following objectives:

- Up to 32 notes can be handled
- The application remains in total control of the Escrow process
- Once a note is in the extended escrow system it is only returned to a user under application control.
- If escrowed notes are returned to a user, only those note supplied by the user are returned automatically.
- Where escrowed notes match note recycler dispensers, the notes are tracked and directed to those dispensers.
- If an application has normal Escrow turned on, it can control the entry of notes into the extended escrow system.
- A note recycler with an extended escrow dispenser defined cannot be enabled using the normal control facilities (As the notes will go to the extended escrow dispenser from where they will unrecoverable.) Note that individual notes can still be enabled and disabled.

Under abnormal conditions, the application decides on the appropriate action, rather than Paylink automatically. Such conditions include:

- System start-up where an escrow operation was in progress.
- Notes “accidentally” directed to the cashbox rather than a recycler.

Functionality

A note recycler is regard by Paylink as being made up of an Acceptor and a number of recycling units.

In general, these recycling units will have been published by Paylink as Dispensers and can be used to make payouts to users.

The pre-requisite for Extended Escrow is that the note recycler has at least one recycling unit that will accept every denomination of note and that has ability to either transfer these notes to the acceptors stacker, or to return them to the user. This is called the escrow recycling unit in this document.

The essential feature of the Paylink Extended Escrow system is that notes are initially accepted into a logical escrow, with the Paylink unit keeping track of which notes have been inserted and where they are being stored.

The application, which uses the Paylink escrow facilities to control and monitor this, can then decide to either stack (keep) the notes, or can decide to return the notes to the user.

The code running within Paylink is responsible for tracking the notes and issuing the appropriate commands to the note acceptor.

Accepting Notes

When the `EXT_ESCROW_ACCEPT` command is issued, notes are accepted to the recycling unit(s) and the `EscrowNote` array is filled in, to detail which notes have been accepted and which recycling unit they are being stored on.

Returning Notes

If an `EXT_ESCROW_RETURN` command is issued, then for every recycling unit Paylink issues a “pay” command to the recycler for the number of notes stored on that unit during the accept phase - thereby returning to the user the notes they have just inserted.

Keeping Notes

If an `EXT_ESCROW_STACK` command is issued, then the application wishes to keep the notes, and the processing varies depending upon which notes have been stored and which recycling units they are store on.

For an escrow-recycling unit that has been configured to be “pure escrow” the Paylink code issues a dump command to transfer all the notes in that unit to the cashbox.

With recycler like the Crane PS B2B300, notes destined for a normal recycling unit that becomes full are just redirected to the escrow-recycling unit. When an `EXT_ESCROW_STACK` command is issued for a device like this Paylink updates its internal level of payable notes, as the unit itself will essentially handle the situation where the unit becomes full.

For an escrow-recycling unit that is configured to also be published as a Dispenser, the Paylink code has to examine all the notes that are in the recycling unit and to compare them with the routing in the corresponding acceptor. Where the note denomination matches the notes is stepped over and kept for subsequent use in payout operations.

Where the note denomination doesn't match, the Paylink code issues a partial dump command to cause the non-matching note and all notes that were inserted later to be stacked to the cashbox.

As a final check, the total number notes being retained on the escrow recycling unit is compared with a set level and if over this level, then the a partial dump command to stack the excess notes to the cashbox has to issued so as to retain enough space to store future escrow notes.

Operation

The operation of the Extended Escrow is quite simple. The Application can at all times discover the state of the escrow system by calling **ReadEscrowBlock**.

The current state of the system is reported in the **State** field.

To control the Escrow system, the application should call **EscrowCommand** with an appropriate parameter. The Escrow system indicates that it has accepted (and is processing) the command by setting the **Result** field to **EXT_ESCROW_COMPLETE**.

For each Escrow system state, the acceptable commands and their results are shown in this table: (for ease of layout, the EXT_ESCROW_ prefix for all the states / commands has been omitted)

Current State	Allowable Commands / Events	New State	Comments
NONE	N/A		No escrow system configured
OFF	START	IDLE	
	START	RETURNED_PROBLEM	If there were notes stored
IDLE	ACCEPT	WAITING	No notes have yet been read
	STOP	OFF	
WAITING	Note being read	LOADING	
	PAUSE	IDLE	
LOADING	Read completed	STORED	This repeats for each note
STORED	Note being read	LOADING	
	Note read that fills the system	FULL	At this point, the acceptor is disabled.
	PAUSE	PAUSED	At this point, the acceptor is disabled.
PAUSED	ACCEPT	LOADING	Acceptance is restarted
	STACK	STACKING	
	RETURN	RETURNING	
STACKING	Stacking OK	STACKED_OK	
	Stacking problem	STACKED_PROBLEM	
RETURNING	Returning OK	RETURNED_OK	
	Returning problem	RETURNED_PROBLEM	
STACKED_OK	ACCEPT	WAITING	This clears the previous transaction
	STOP	OFF	
STACKING_PROBLEM	Problem fixed	STACKED_OK	This status stays until fixed
RETURNED_OK	ACCEPT	WAITING	This clears the previous transaction
	STOP	OFF	
RETURNING_PROBLEM	RETURN	RETURNING	Retry the return
FULL	PAUSE	PAUSED	

Abnormal Situations

If there are notes in the extended escrow dispenser at start up, they need to be processed under the control of the Application. To allow the application to detect this, there are two abnormal states that can be presented when the extended escrow system transitions from `OFF` when the `START` command is issued. Either `RETURNED_PROBLEM` which indicates that the notes were actively being returned by Paylink when it last ran, or `POWER_ACTIVE` which indicates that the notes were being held and had not yet been processed.

Cashless Processing

Background

There are a number of cashless systems available that are supported by Paylink. These include: Credit Card Acceptance and Ticket In / Out applications for AWP systems.

Paylink provides an interface to support this for acquiring credit (from a ticket or from a cashless device) and returning it to a cashless device based system if the peripheral supports that. The `MaximumPay` field in the control object operates as a flag for this; if it is non-zero then credit payment is possible

In all cases, the remote processing runs autonomously, and Paylink provides a Cashless Object that reflects the processing of the remote unit, which should be regularly polled by application in order to determine what is happening.

Data is transferred to and from the Paylink system using this Cashless Object, and control of the process is via a number of functions, whose names all start `Cashless...`

The most import property of the Cashless object is the current state, which allows the application to see where the cashless system is up to.

The Values for the `CurrentState` item are duplicated from the API manual here:

Name	Value	
CR_NO_UNIT	0	No appropriate unit connected
CR_BUSY	1	Busy
CR_DISABLED	2	Idle and disabled
CR_IDLE	3	Idle
CR_FAULT_DISABLED	4	The device has become non operational
CR_FAULT_IDLE	5	The device has become non operational
CR_AVAILABLE_IDLE	6	Idle, but prepared for Credit Requests
CR_INVALID_OP	100	Invalid operation
These are only valid during Credit Input operations		
CR_AVAILABLE	11	Arbitrary Credit Available
CR_CONFIRMED	12	Credit Request from Application Accepted
CR_TAKEN	13	Credit Actually Taken for remote source
CR_REFUSED	14	Credit Request from Application Refused
CR_CANCELLED	15	Application has successfully cancelled the transaction
CR_FAILED	16	The device failed during processing - no credit taken
CR_FAILED_TAKEN	17	The device failed during processing - credit was taken
CR_INVALID_REF	101	Invalid Reference
These are only valid during Credit Payment operations		
CR_TRANSFERRING	21	A credit output operation is in progress
CR_TRANSFERED	22	The Credit has been accepted by the device
CR_TRANSFER_FAIL	23	The credit has been refused by the device

Processing

Credit Card Sequencing

Note that for each function `CurrentState` will normally return the transient state `CR_BUSY` before becoming the state shown in the table.

Credit Input Sequencing

The following table shows the normal sequence of states taken by `CurrentState` during a typical credit card transaction:

State	Normally Entered Because:
<code>CR_NO_UNIT</code>	System Startup
<code>CR_DISABLED</code>	Paylink discovers the cashless unit
<code>CR_IDLE</code>	Application Calls CashlessEnable or CashlessReset
<code>CR_AVAILABLE</code>	Credit Card Presented
<code>CR_AVAILABLE_IDLE</code>	System idle with e.g. MDB Level 3 "always Idle" processing enabled.
<code>CR_CONFIRMED</code>	Application Calls CashlessRequestCredit and the credit is available.
<code>CR_TAKEN</code>	Application Calls CashlessTakeCredit
<code>CR_REFUSED</code>	Application Calls CashlessRequestCredit and the credit is not available.
<code>CR_CANCELLED</code>	Application Calls CashlessCancel

The State "`CR_AVAILABLE_IDLE`" should in general be processed in exactly the same way as "`CR_AVAILABLE`", the only difference being that it is the normal state of the unit and the value in `CreditValue` has been specified in the configuration file, not read from the peripheral.

Credit Output Sequencing

The following table shows the normal sequence of states taken by `CurrentState` during a typical credit card revalue transaction:

State	Normally Entered Because:
<code>CR_NO_UNIT</code>	System Startup
<code>CR_DISABLED</code>	Paylink discovers the cashless unit
<code>CR_IDLE</code>	Application Calls CashlessEnable or CashlessReset
<code>CR_AVAILABLE</code>	Credit Card Presented
<code>CR_AVAILABLE_IDLE</code>	System idle with e.g. MDB Level 3 "always Idle" processing enabled.
<code>CR_TRANSFERRING</code>	Application Calls CashlessPayCredit and the credit is acceptable.
<code>CR_TRANSFERED</code>	The peripheral device has accepted the credit
<code>CR_TRANSFER_FAIL</code>	The valid request has not completed (Invalid requests result in a <code>CR_INVALID_OP</code> status)

Ticket Sequencing

The following table shows the normal sequence of states taken by CurrentState during a typical ticket transaction:

State	Normally Entered Because:
CR_NO_UNIT	System Startup
CR_IDLE	Paylink discovers the cashless unit or CashlessReset
CR_AVAILABLE	Application calls SubmitTicket with a valid ticket reference
CR_CONFIRMED	Application Calls CashlessRequestCredit for the value on the ticket.
CR_TAKEN	Application Calls CashlessTakeCredit
CR_REFUSED	Application Calls CashlessRequestCredit and the credit is not available.
CR_CANCELLED	Application Calls CashlessCancel

Abnormal Processing

The above table represent the normal way that credit acquisition proceeds. Paylink however includes facilities to handle abnormal situations. These situations use the fields: `TotalAcquisitions` and `TotalCredit`. These are updated in real time and preserved over a power fail situation. An application that wishes to handle such situations can store the values of these before starting a transaction, and then check them on startup.

MDB Cashless processing

The most common peripherals that is used with Paylink cashless processing is a standard MDB cashless peripheral.

The MDB standard allows for different modes of operation: the original standard one and a new contactless optimised one.

With the **original standard processing** a transaction starts with a token or card being presented to the peripheral. The peripheral validates the token / card and then informs Paylink of the availability of a certain amount of credit. This corresponds to the `CR_AVAILABLE` status, with this amount shown in the control block. The application can *then* ask the user what they want to purchase.

With the **contactless optimised processing** the peripheral is permanently in the state where it will perform a credit transfer, the transaction starts with the application, which sends an unsolicited request to the peripheral, which then fails if the amount is too high. This mode is known in the MDB documentation as “always idle”.

In use, the mode that will be used is determined by Paylink and is set in the configuration file. The basic entry of

Cashless at 10h

sets Paylink up to use the original standard processing, whereas

Cashless at 10h idle 3000

sets Paylink up to use the contactless optimised processing. The value following the idle keyword is presented to the application as the available credit, but is not relevant to any of the processing done by Paylink.

MDB Cashless Credit

The MDB cashless specification allows for credit transfer from the Paylink unit to a token / card presented to the peripheral. The MDB Cashless messages provide two separate routes to transfer credit to the MDB peripheral: Negative Vend and Revalue.

It is not explicitly stated in the protocol, but Revalue is most appropriate for standard processing and Negative Vend for the contactless optimised “always idle”.

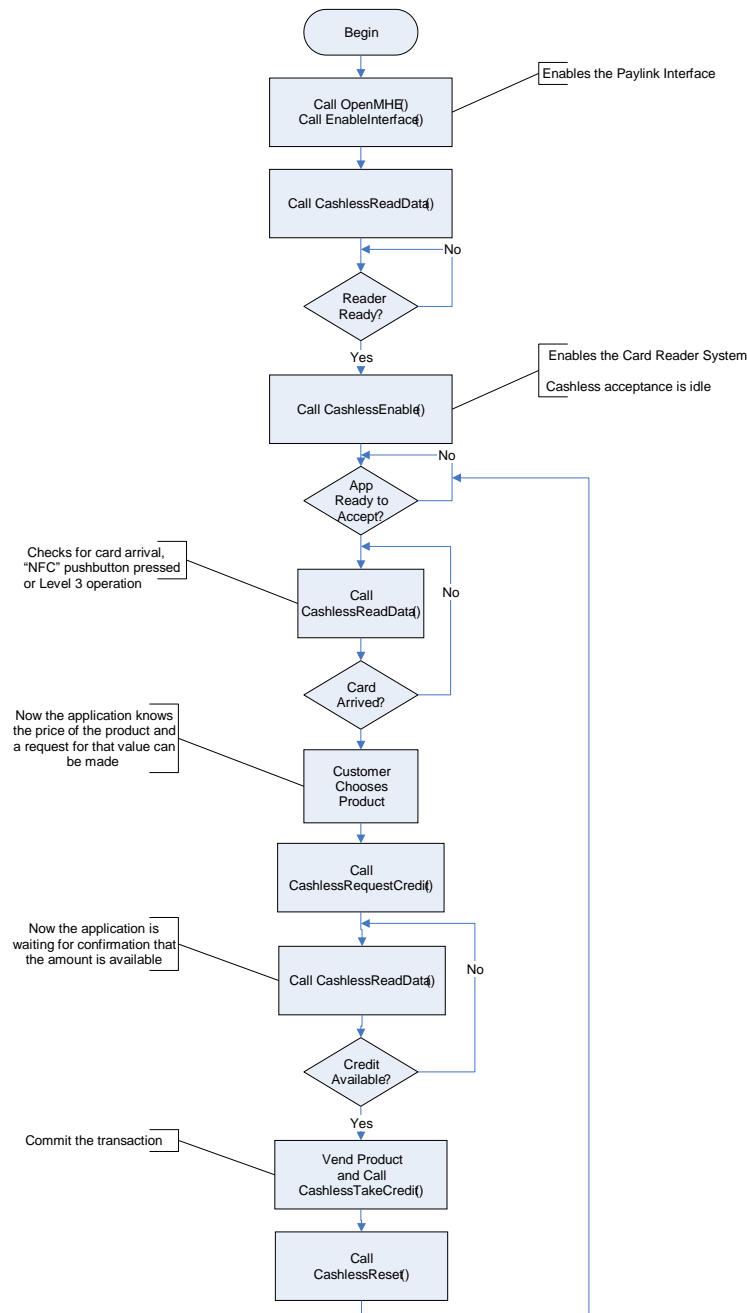
For the standard mode, when a session starts the peripheral is queried for the maximum value of a Revalue request and the reply number set in **MaximumPay**.

For the “always idle” mode, if the “Negative Vend” available flag is set then **MaximumPay** is reported as 65535

When **CashlessPayCredit** is called, then if the value is below the Revalue limit or Negative Vend is not available then a Revalue command is issued, otherwise a Negative Vend command is used.

Example Cashless Transaction

The following diagram shows a typical cashless transaction for a vending application and a card reader.



Meters / Counters

The Standard Paylink units support the concept of external meters that are accessible from the outside of the PC system.

In keeping with the Paylink concept, an interface is defined to an idealised meter. This will be implemented transparently by the card using the available hardware. Currently the Paylink unit will support either a ***Starpoint Electronic Counter***, or from 1 to 8 mechanical meters.

Mechanical Meters (1.12.4)

From 1.12.4 onwards, Paylink supports mechanical meters, driven using pulses through the general-purpose high power outputs. Suitable meters are required to operate on DC at 20 pulses per second or faster.

Configuration file entries are used to map Counter Numbers 1 to 8 onto the Paylink outputs.

Paylink records how many pulses have been sent, and how many are currently required. It attempts to handle the fact that while the pulses are being output the power may be cycled. Paylink updates its non-volatile memory as it turns on the transistor at the start of the pulse. This means that during a power cycle, at most one pulse may be lost (as it is not driven for long enough) but no spurious pulses can be generated.

Events (Faults / Auditing)

Introduction

The Paylink system design is based around the handling of money in and money out. The peripherals used are also capable of providing notifications that are not related to money, primarily faults and fraud attempts.

This information is captured by Paylink, and is provided to the application as “Events”, which are queued and passed to the application on request.

There is no intention that these events would be used for the normal operation of the application. Rather, the intention is that they can be captured and presented in “management” reports.

(Obviously, the application can respond automatically to events such as fraud, by disabling everything for a while, but this doesn’t form part of the algorithms by which the application manages the peripherals.)

The event codes used have an internal structure, allowing categorizations. The bottom 6 bits are the unique event classification code, fault related codes have bit 5 set and otherwise overlap these events code, whilst more significant bits describe the type of unit affected.

For details of the exact makeup of the values of these codes, users are referred to the `ImheiEvent.h` header file.

Events fall into two categories, notifications and faults. Notifications are just that, the incoming information is passed along to the application.

On the other hand, Paylink remembers the fact of a fault having happened, and when the fault clears, a `NOW_OK` “fault” event will be generated.

A specific bit in the event code is reserved for indicating fault events.

Full details on the make up of the event codes are given in the “Milan / Paylink Programmers Manual” document.

cctalk coin processing

Fault Events

During start-up the cctalk command “Do Self Test” is sent to the acceptor. The response is queued as an event with the first byte of the response in **RawEvent** and an **EventCode** type of **IMHEI_COIN_NOW_OK** or **IMHEI_COIN_UNIT_REPORTED_FAULT**.

If the unit is reset (the sequence number is found to be zero) or repeated messages are ignored **IMHEI_COIN_UNIT_RESET** or **IMHEI_COIN_UNIT_TIMEOUT** event is queued. Whenever any of these faults have been reported, the handler will continually “poll” the acceptor with “Do Self Test” commands until a “non-faulty” response is returned.

Coin Events

When the acceptor reports an event other than an accepted coin, this is queued as a **COIN_DISPENSER_EVENT** event, with the actual event byte reported in **RawEvent**.

The events categorised as **OUTPUT_PROBLEM**, **JAM** & **INTERNAL_PROBLEM**, are also reported as self-test faults on some acceptors. They are therefore automatically latched as faults (without sending the self-test fault) and hence a **NOW_OK** “fault” is generated when they clear.

The handler classifies cctalk events as:

Event Number	Meaning	Event Classification
1	Coin Rejected	REJECTED
2	Coin Inhibited	INHIBITED
3	Multiple window	REJECTED
4	Wake-up timeout	JAM
5	Validation timeout	JAM
6	Credit sensor timeout	JAM
7	Sorter opto timeout	OUTPUT_PROBLEM
8	2nd close coin error	REJECTED
9	Accept gate not ready	REJECTED
10	Credit sensor not ready	REJECTED
11	Sorter not ready	REJECTED
12	Reject coin not cleared	REJECTED
13	Validation sensor not ready	REJECTED
14	Credit sensor blocked	JAM
15	Sorter opto blocked	OUTPUT_PROBLEM
16	Credit sequence error	FRAUD
17	Coin going backwards	FRAUD
18	Coin too fast (over credit sensor)	FRAUD
19	Coin too slow (over credit sensor)	FRAUD
20	C.O.S. mechanism activated (coin-on-string)	FRAUD
21	DCE opto timeout	FRAUD
22	DCE opto not seen	FRAUD
23	Credit sensor reached too early	FRAUD
24	Reject coin (repeated sequential trip)	FRAUD
25	Reject slug	FRAUD
26	Reject sensor blocked	JAM
27	Games overload	INTERNAL_PROBLEM
28	Max. coin meter pulses exceeded	INTERNAL_PROBLEM
128-159	Inhibited Coin	INHIBITED
254	Flight Deck Open	RETURN

cctalk note processing

Fault Events

Shortly after start-up the cctalk command “Do Self Test” is sent to the acceptor. The response is queued as an event with the first byte of the response in **RawEvent** and an **EventCode** type of **IMHEI_NOTE_NOW_OK** or **IMHEI_NOTE_UNIT_REPORTED_FAULT**.

Some acceptors reply to this command with a NAK, these are reported as **IMHEI_NOTE_SELF_TEST_REFUSED**.

If the unit is reset (the sequence number is found to be zero) or repeated messages are ignored **IMHEI_NOTE_UNIT_RESET** or **IMHEI_NOTE_UNIT_TIMEOUT** event is queued.

Whenever any of these faults have been reported, the handler will continually “poll” the acceptor with “Do Self Test” commands until a “non-faulty” response is returned.

Note Events

When the acceptor reports an event other than an accepted note, this is queued as an **NOTE_DISPENSER_EVENT** event, with the actual event byte reported in **RawEvent**.

The events categorised as **MISAREAD**, **JAM** & **INTERNAL_PROBLEM**, are also reported as self-test faults on some acceptors. They are therefore automatically latched as faults (without sending the self-test fault) and hence a **NOW_OK** “fault” is generated when they clear.

The handler classifies cctalk events as:

Event Number	Meaning	Event Classification
0	Master inhibit active	INHIBITED
1	Bill returned from escrow	RETURN
2	Invalid bill (due to validation fail)	REJECTED
3	Invalid bill (due to transport problem)	REJECTED
4	Inhibited bill (on serial)	INHIBITED
5	Inhibited bill (on DIP switches)	INHIBITED
6	Bill jammed in transport (unsafe mode)	MISREAD
7	Bill jammed in stacker	OUTPUT_PROBLEM
8	Bill pulled backwards	FRAUD
9	Bill tamper	FRAUD
10	Stacker OK	OUTPUT_FIXED
11	Stacker removed	OUTPUT_PROBLEM
12	Stacker inserted	OUTPUT_FIXED
13	Stacker faulty	OUTPUT_PROBLEM
14	Stacker full	OUTPUT_PROBLEM
15	Stacker jammed	OUTPUT_PROBLEM
16	Bill jammed in transport (safe mode)	JAM
17	Opto fraud detected	FRAUD
18	String fraud detected	FRAUD
19	Anti-string mechanism faulty	INTERNAL_PROBLEM

***cctalk* hopper processing**

This is divided into two parts, the processing associate with reporting the ongoing ability of a functioning hopper to pay out coins, and that associated with checking that the hopper is operational.

Both of these require a “Test Hopper” command to be sent to the unit, but the reporting mechanism is different.

The ongoing ability to pay out is reported as the Status field in the dispenser block, the results of the regular check are reported as “self-test” events.

Note: that when a Payout is issued the results of any “Self Test” are ignored - the dispense coins command is sent to the hopper regardless.

On a regular basis the “Test Hopper” command is sent to the each hopper when otherwise idle, and the result evaluated. After start-up, and regularly thereafter, a `IMHEI_COIN_DISPENSER_NOW_OK` is reported if there are no errors.

The defined return from this command is a string of up to 4 bytes (depending upon the exact unit) with one (or theoretically more) bits set to indicate the problem.

The action of Paylink is to regard these bytes as containing 32 bits. The bits are classified by this section of Paylink as an Error, a Fraud attempt, a Payout result or “information only”. Paylink scans along these bits looking for the first Error or Fraud bit that is non-zero. Other bits are ignored.

The bit number of this first bit (i.e. a number in the range 0 to 31) is then returned in **RawEvent** and an **EventCode** of either `IMHEI_COIN_DISPENSER_FRAUD_ATTEMPT` or `IMHEI_COIN_DISPENSER_REPORTED_FAULT`

For reference, the bit numbers, and their classification are:

Bit Number	Meaning	Event Classification	Payout Result
0	Jammed	Information only	PAY_JAMMED
1	Empty	Information only	PAY_EMPTY
2	Reversed	Information only	
3	Idle fraud blocked	Fraud	PAY_FRAUD
4	Idle fraud short	Fraud	PAY_FRAUD
5	Payout blocked	Information only	PAY_FAILED_BLOCKED
6	Power up	Information only	
7	Disabled	Fault	
8	Fraud short	Fraud	PAY_FRAUD
9	Sngle coin mode	Fault	
10	Chksum a	Fault	
11	Chksum b	Fault	
12	Chksum c	Fault	
13	Chksum d	Fault	
14	Pwr fail during write	Fault	
15	Pin locked	Fault	
16	Powerdown during payout	Information only	
17	Unknown coin type paid	Fault	
18	Pin number incorrect	Fault	
19	Incorrect cipher key	Fault	
20	Unused	Information only	
21	Unused	Information only	
22	Unused	Information only	
23	Unused	Information only	
24	Unused	Information only	
25	Unused	Information only	
26	Unused	Information only	
27	Unused	Information only	
28	Unused	Information only	
29	Unused	Information only	
30	Use other hopper	Information only	PAY_NOT_EXACT
31	Opto fraud	Fraud	PAY_FRAUD

ID-003 note processing

Fault Events

There is no specific self-test command with ID-003, the acceptor reports faults in response to a poll. When the protocol handler completes its initialisation, the first idle response is reported as **IMHEI_NOTE_NOW_OK**.

When a **FAILURE** response to a status poll is received, this is reported as an **IMHEI_NOTE_UNIT_REPORTED_FAULT** event. A failure status is expected to be continually reported by the acceptor until it is cleared. When the acceptor again reports **IDLING**, then an **IMHEI_NOTE_NOW_OK** event is reported.

Other “non normal” responses to a status poll are reported as events as they are receive according to the table below.

In a similar way to the action for faults, **OUTPUT_FIXED** is reported when events that translate to **OUTPUT_PROBLEM** are cleared.

Status Value	Name	Event Classification
0x17	REJECTING	REJECTED
0x41	POWER UP WITH BILL IN ACCEPTOR	REJECTED
0x42	POWER UP WITH BILL IN STACKER	REJECTED
0x43	STACKER FULL	OUTPUT_PROBLEM
0x44	STACKER OPEN	OUTPUT_PROBLEM
0x45	JAM IN ACCEPTOR	JAM
0x46	JAM IN STACKER	OUTPUT_PROBLEM
0x47	PAUSE	UNKNOWN
0x48	CHEATED	FRAUD
0x49	FAILURE	- Fault Report
0x4A	COMMUNICATION ERROR	INTERNAL_PROBLEM

CCNet note processing

Fault Events

There is no specific self test command with CCNet, the acceptor reports faults in response to a poll. When the protocol handler completes its initialisation, the first idle response is reported as **IMHEI_NOTE_NOW_OK**.

When a **FAILURE** response to a status poll is received, this is reported as an **IMHEI_NOTE_UNIT_REPORTED_FAULT** event. A failure status is expected to be continually reported by the acceptor until it is cleared. When the acceptor again reports **IDLING**, then an **IMHEI_NOTE_NOW_OK** event is reported.

Other “non normal” responses to a status poll are reported as events as they are receive according to the table below.

In a similar way to the action for faults, **OUTPUT_FIXED** is reported when events that translate to **OUTPUT_PROBLEM** are cleared.

Most status values are part of the normal running of the system, the following statuses are regardless as reporting unusual / fault events and are reported through the event system.

Status Value	Name	Event Classification	RawEvent
0x1C 0x68	REJECTING	INHIBITED	0x68
0x1C nn	REJECTING	REJECTED	nn
0x41	DROP CASSETTE FULL	OUTPUT_PROBLEM	0x41
0x42	DROP CASSETTE REMOVED	OUTPUT_PROBLEM	0x42
0x43	JAM IN ACCEPTOR	MISREAD	0x43
0x44	JAM IN STACKER	OUTPUT_PROBLEM	0x44
0x45	CHEATED	FRAUD	0x45
0x47 nn	GENERIC BB ERROR	FAULT	nn
0x82 nn	RETURNED	RETURN	nn
0x14	IDLING	Generate an OK_NOW or OUTPUT_FIXED if in fault / output problem.	
0x19	DISABLED		

EBDS (SC/SCR) note processing

Fault Events

Not all EBDS acceptors support self-test. Where they do, a self-test fail is reported with the index of the first of the 20 integers that has reported an error.

When self-test does not report any error, and when the protocol handler completes its initialisation, the first idle response is reported as **IMHEI_NOTE_NOW_OK**.

When a status poll response is received with fault bits set, the bit numbers that are set are reported in the RawEvent data field in one or more independent event.

Bit Number	Name	Reported as
0	EBDS_CHEATED	IMHEI_NOTE_FRAUD_ATTEMPT
1	EBDS_REJECTED	IMHEI_NOTE_REJECT_NOTE
2	EBDS_JAMMED	IMHEI_NOTE_ACCEPTOR_JAM
3	EBDS_STACKER_FULL	IMHEI_NOTE_STACKER_PROBLEM
4	EBDS_CASSETTE_MISSING	IMHEI_NOTE_STACKER_PROBLEM
5	EBDS_PAUSED	IMHEI_NOTE_UNCLASSIFIED_EVENT
6	EBDS_CALIBRATING	IMHEI_NOTE_UNCLASSIFIED_EVENT
9	EBDS_INVALID_COMMAND	IMHEI_NOTE_UNCLASSIFIED_EVENT
10	EBDS_FAILURE	IMHEI_NOTE_UNCLASSIFIED_EVENT
14	EBDS_TRANSPORT_OPEN	IMHEI_NOTE_STACKER_PROBLEM

For events reported as IMHEI_NOTE_STACKER_PROBLEM an event of IMHEI_NOTE_STACKER_FIXED will be reported when the bit clears. For other bits an event of IMHEI_NOTE_NOW_OK will be reported when the bit clears.

BCR / CR10x Fault Processing

The current Paylink API design only allows for a single byte of error data. The BCR / CR10x devices provide 3 bytes, and so to allow for this the initial fault report is followed by an additional information byte. The contents of these two bytes are described here:

This input to this description is a merged list of error codes from the documents:
CR100 TSP182 Issue 0.9.1 and Bulk Coin Recycler TSP151 Issue 3.8

All error codes will **always** be reported as:

- EventCode = **IMHEI_COIN_UNIT_REPORTED_FAULT**
- RawEvent = Value from Code column
- Index = BCR Acceptor

In addition, those error codes marked in blue below will be reported with a subsequent event as:

- Event Code = **IMHEI_COIN_DISPENSER_REPORTED_FAULT**
- RawEvent = Optional Extra Info for Code 26
Optional Extra Info + 100 for Code 27
- Index = Corresponding BCR Hopper

and those error codes marked in Yellow below will be reported with a subsequent event as:

- Event Code = **IMHEI_COIN_INTERNAL_PROBLEM**
- RawEvent = Value from Last Column + Optional Extra Info

Code	Fault	Optional Extra Info	Only	Stat 1	Stat 2	Yellow Offset
0	OK (no fault detected)	0		-	0	
1	EEPROM checksum corrupted	1 = Coin acceptor checksum error		100	103	
		2 = Controller checksum error	CR	255	255	
2	Fault on inductive coils	1 to 5 = Validation coil		100	103	
		6 = Singulator belt sensor (sensor missing)		255	255	
		7 = Coin return sensor (sensor missing)		255	255	
		8 = Singulator belt sensor (active but no belt move)		255	255	
		9 = Coin return sensor (active but no belt move)		255	255	
		10 = Coin acceptor wake-up	CR	100	103	
3	Fault on credit sensor	0		100	103	
8	Fault on sorter exit sensors	1 = 4-way sorter (blocked)	BCR	100	101	
		2 = 8-way diverter (blocked)	BCR	100	101	
		3 = 8-way diverter (timeout, coin not seen)	BCR	100	104	
		11 to 18 = Carousel gate	CR	255	255	
9	NVRAM checksum corrupted	0		255	255	
19	Fault on coin return mechanism	0		100	101	
22	Fault on thermistor	0		100	103	

Code	Fault	Optional Extra Info	Only	Stat 1	Stat 2	Yellow Offset
23	Payout motor fault	1 = Singulator (jammed)		255	255	0
		2 = Escalator (jammed)	BCR	255	255	0
		2 = Conveyor (jammed)	CR	255	255	0
		3 = Motorised reject		255	255	0
		4 = Singulator (no tach)		255	255	0
		5 = Escalator (no tach)	BCR	255	255	0
		5 = Conveyor (no tachs)	CR	255	255	0
		6 = Sing. (sensor block)		255	10	0
		7 = Esc. (sensor block)		255	11	0
		8 = Carousel (no tachs)	CR	255	255	0
		9 = Carousel (jammed)	CR	255	255	0
		10 = Coin rotor – fault	CR	100	103	0
		11 = Coin rotor – tachs	CR	100	103	0
		12 = Coin rotor – diameter opto missing	CR	100	103	0
		13 = Coin rotor – park opto missing	CR	100	103	0
		14 = Coin rotor – coin jam	CR	100	101	0
		15 = Coin rotor – excessive jam-jogs	CR	100	101	0
16 = Singulator belt broken	BCR	255	255	0		
17 = Escalator belt broken	BCR	255	255	0		
26	Payout sensor fault	Hopper = 1 to 8 Opto blocked payout		1to8	2	
		Hopper = 11 to 18 Opto blocked idle		1to8	3	
		Hopper = 21 to 28 Opto short-circuit idle		1to8	3	
		Hopper = 31 to 38 Opto short-circuit payout		1to8	3	
		Hopper = 41 to 48 Max. current exceeded		1to8	4	
		Hopper = 51 to 58 Bad EEPROM checksum		1to8	4	
		Hopper = 61 to 68 Power fail on write		1to8	4	
27	Level sensor error	1 = Hopper 1		1	4	
		2 = Hopper 2		2	4	
		3 = Hopper 3		3	4	
		4 = Hopper 4		4	4	
		5 = Hopper 5		5	4	
		6 = Hopper 6		6	4	
		7 = Hopper 7	CR	7	4	
		8 = Hopper 8	CR	8	4	
32	Internal comms bad	0 (parallel interface to validator)	CR	255	255	
33	Supply voltage outside operating limits	1 = +24V rail		255	255	
		2 = +5V rail		255	255	
35	D.C.E. fault	0, 1 = Blocked		?	100	
		2 = Broken		100	103	
40	RAM test fail	0		255	255	

Code	Fault	Optional Extra Info	Only	Stat 1	Stat 2	Yellow Offset
48	Slave device not responding	1 = Cashbox missing		250	252	20
		2 = Hopper tray missing		255	255	20
		3 = No hoppers fitted		255	255	20
		4 = Hopper ID mis-match		255	255	20
		5 = Coin acceptor missing		103	1	20
		6 = No hop. lowest coin		255	255	20
		7 = Mixed currency CA		255	255	20
		8 = Mixed currency HO		255	255	20
		9 = I2C communication error	CR	255	255	20
49	Fault on opto sensor	1 = Reserved		-	-	40
		2 = Cashbox full sensor		250	251	40
		3 = Wake-up sensor (Avalanche sensor)		255	255	40
		4 = Exit cup full sensor		255	255	40
		5 = Accept flap	CR	255	255	40
		6 = Cashbox flap	CR	255	255	40
		7 = Carousel track opto	CR	255	255	40
		8 = Carousel pusher opto (credit sensor)	CR	255	255	40
		9 = Coin acceptor diameter opto	CR	100	103	40
		10 = Coin acceptor park opto	CR	100	103	40
50	Battery fault	0		255	255	
51	Door open	1 = Singulator door		255	255	60
		2 = Escalator door	BCR	255	255	60
		3 = Carousel lid	CR	255	255	60
52	Microswitch fault	1 = Reject home		255	255	
53	RTC fault	1 = Read / write fail		255	255	
54	Firmware error	1 = CA bad firmware ID		255	255	
		2 = CA firmware too old		255	255	
		3 = HO bad firmware ID		255	255	
		4 = HO firmware too old		255	255	
		5 = HO bad build ID		255	255	
		6 = CA bad build ID		255	255	
		7 = Unhandled CA event		255	255	
		8 = CA programming err.		255	255	
		9 = Bad DB hop. config.		255	3to8	
		10 = Prog. checksum err.		255	9	
		11 = Error during upgrade		255	9	
		12 = Stack overflow		255	9	
55	Initialisation error	1 = Flush timeout		255	255	
56	Supply current outside operating limits	1 = +5V rail	CR	255	255	
		2 = Hopper rail	CR	255	255	
		3 = Solenoid rail	CR	255	255	

CLS Fault Processing

All errors are reported against the acceptor device, except for hopper error 0x65 (lost connection) which is reported against the failing hopper.

The current Paylink API design only allows for a single byte of error data. The CLS provides 2 bytes in the event codes.

To allow for this the error byte is encoded when they are reported following the CLS entering an error state.

The reported errors come from two sources, the CLS controller and the acceptor, but can be merged into one list as they do not have overlapping values in the **2nd** byte. (All acceptor faults have a first byte value of 0x20)

The encoding for the single byte in the RawEvent byte is as follows (in hex):

RawEvent Byte	Two byte error code
01 - 03	00 01 - 00 03
04 - 07	00 10 - 00 13
08 - 0B	00 20 - 00 23
0C - 0F	00 30 - 00 33
10 - 6F	20 10 - 20 6F
70 - 7F	11 D0 - 11 DF
B0 - BF	0B 00 - 0B 0F
D0 - DF	0D 00 - 0D 0F
E0 - EF	0E 00 - 0E 0F

To allow the application to confirm the encoding if necessary, an event of **IMHEI_COIN_INTERNAL_PROBLEM** is also queued, with the RawEvent field containing the first of the two error bytes.

F53/F56 Fault Processing

The F56 is complex mechanically, so a jam situation is reported as an error for a number of specific reasons, which are reported via the Event mechanism.

The F56 handler generates events to notify the application of these jam situations. Specifically, the handler will return an

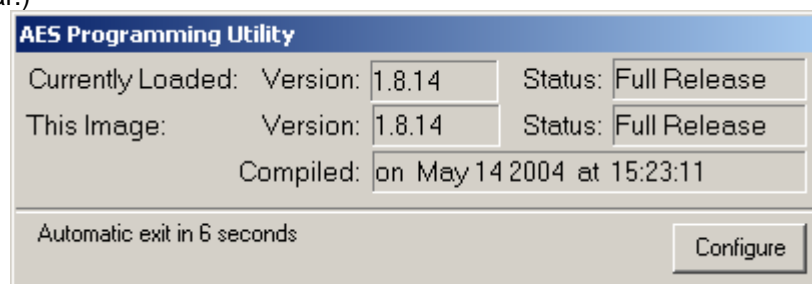
- EvenCode = **IMHEI_NOTE_DISPENSER_PROBLEM**
- RawEvent = the returned F56 error code (as described in the Fujitsu “F56-BDU ERROR CODE LIST” manual.)
- Index = F56 Cassette.

Firmware reprogramming

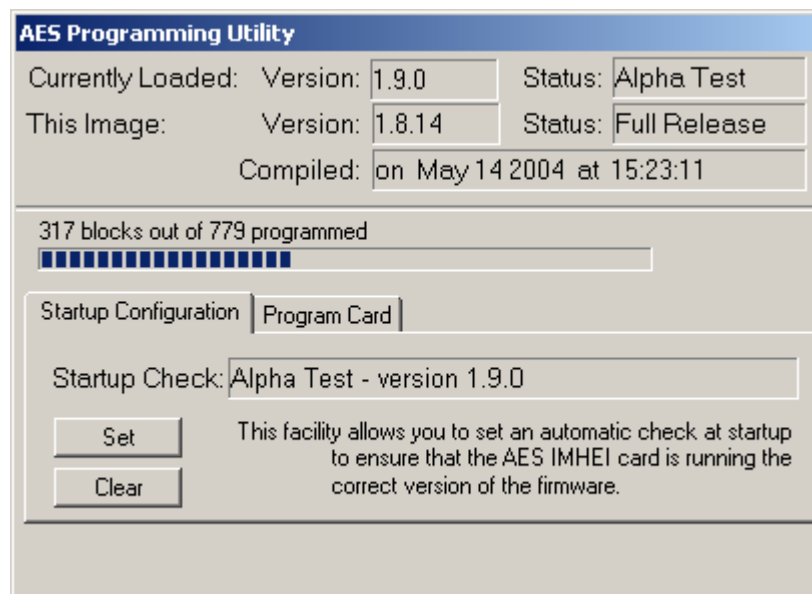
All firmware releases for Standard Paylink are distributed as self-extracting Windows executables. As well as those contained in the distribution, they are also accessible individually on the Internet at: <http://www.aardvark.eu.com/products/milan/downloads.htm>. as zip files Each executable name includes Vx-x-x-x, where the sequence after the V identifies the version as described at the start.

The same Windows programming utility is contained in all the firmware release files. When run normally it will check that a Standard Paylink unit is installed and accessible, and will then compare the version of the firmware that it contains with version installed on the Interface. If they differ it will then load the new firmware:

if they are the same it will display the (matching) details for 10 seconds and then automatically exit. (If a parameter is provided on the command, then the check is silent if it passes; the waiting display does not appear.)

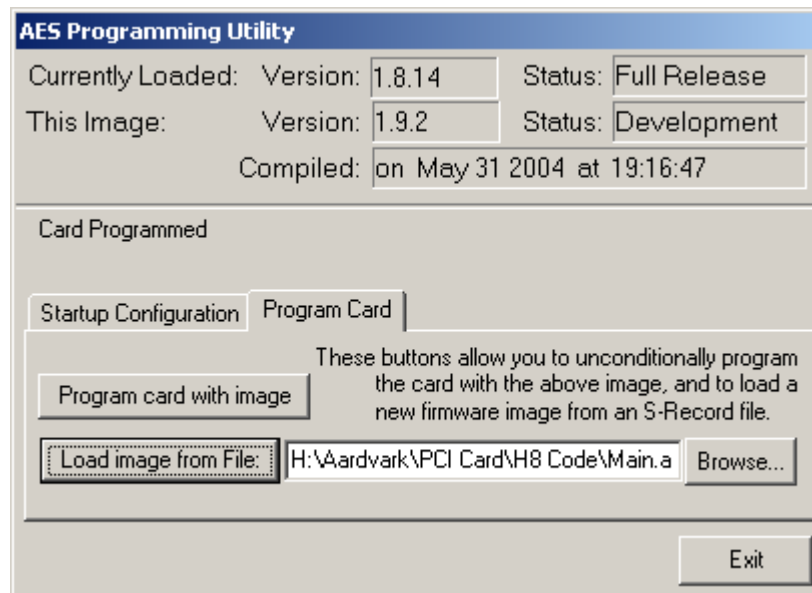


While running a “Configure” button is accessible. This can be used to access two advanced features, “Startup Checking” and “Programming”.



“**Startup Configuration**” provides the ability to “Set” and “Clear” an entry in the Windows registry that will silently run this copy of the programming utility at system Startup. If this entry is Set, it ensures that, if a unit is installed with a different version, the firmware packaged with this copy of the programming utility is loaded onto it.

Note: As the Set entry is for the programming utility itself, in order to use this facility the programming utility must have been saved to a folder on the hard disc and then run from that location.



The **“Program Card”** option will normally only be used by expert users, or under instruction from a support technician. The following two facilities are available:

1. The image packaged with the executable can be overwritten (*for this execution only*) by that contained in a S-Record file.
2. The current image (either packaged or loaded) can be written to the Milan Interface (regardless of whether the version matches).

Command Line Options

Three options are accepted on the command line, the first two are for use primarily from a remote PC.

/Force - will automatically re-program the Milan unit even if the images match.

/Nogui - will never display anything on the screen and will report progress to stdout or a console window, if either is available.

/Check - will cause the loader to exit without showing an window if the Milan firmware matches, and has no errors.

Limitations

A limitation of this programming utility is that a functional release must be executing on the unit. In the event that an earlier or non-functional version is loaded, you will need a special serial cable, the Hitachi programming utility FlashSimple and an S-Record (.a37) file.

For full details on using these, please contact us.

Milan / Paylink Driver Program Configuration

An essential component of the Paylink system is the Driver program, and the method chosen to specify the configuration of the system is to describe the peripheral configuration in a simple text file.

With the standard (metal box) Paylink unit, when contact is established between the Driver program and the unit this file is downloaded to the Milan / Paylink unit as the system starts up.

The unit compares the configuration in the file specified with that stored on the Paylink. If there are any differences, the new configuration is stored and the Paylink unit resets to use the updated configuration.

When a system uses modern Paylink Lite unit(s), or for a purely USB system, the this file configures the code running in the Paylink driver program itself

Driver Parameters

When **Paylink.exe / AESCDriver** is run it needs to find this configuration file. The path to the configuration file can be provided as a single parameter, if no parameter is provided then it will try to read a file called "Standard.cfg" in the folder from which it is run.

Some of the configuration of the driver program itself can be accomplished by two means, with identical results, either by parameters in the configuration or on the command line itself

For **Windows**, switch parameters can be used as follows:

- /S <Paylink Serial>** is used in multiple Paylink installations, and specifies the serial number of the Paylink device that this driver is to connect to.
- /L <Log File Name>** is the full path to the desired log file.
- /Z <K Bytes>** is the maximum size the log file is allowed to reach. If it reaches this size, then it is renamed with a ".old" extension added and a new file started. If omitted, the maximum size is 128K bytes.
- /H** if this is used no window or taskbar Icon is displayed.
- /V** if this is used a normal window is shown on the screen.

For **Linux**, switch parameters can be used as follows:

- s <Paylink Serial>** is used in multiple Paylink installations, and specifies the serial number of the Paylink device that this driver is to connect to.
- p** Run this driver program at a high priority.
(10 less than the SCHED_RR maximum)
- v** Output detailed driver diagnostics to stdout. (See RUN VISIBLE)
- t** (Not for general use) display internal USB link messages.

Multiple Paylink Unit Support.

Although the Paylink system was designed around the idea of a single Paylink unit being connected to a PC, facilities *are* provided to support multiple Paylink units.

The only change that is visible to a programmer when multiple units are in use is that the **OpenSpecificMHE** is used to associate the program with a specific one of the multiple Paylink unit interface areas.

It is envisaged that in a system with multiple Paylink units a separate instance of the program will be running for each Paylink unit interface area and a supervisory level will start the different programs. This is not compulsory as **OpenSpecificMHE** can be called repeatedly with different parameters so as to switch between Paylink unit interface areas.

Unit Identification

The USB interface chip on a Paylink unit provides a “Serial Number”. This is pre-set during manufacture to AE000001 - but is not used or checked in a system that does not have multiple units.

When the **Paylink / AECDriver** program is run, the default is for it to search *all* USB devices that may be a Paylink, and connect to the first one it finds. When the `/S=<SerialNo>` switch is provided on the command line, this has two effects:

Firstly, it causes the driver program to create a named Paylink unit interface area, which can then be connected to by an **OpenSpecificMHE** call with a matching parameter.

Secondly it causes the driver program to search all USB devices that may be a Paylink until it finds one with a matching programmed serial number.

The serial number is *not* associated with the Paylink firmware, and any release of Paylink firmware may be used in a multiple Paylink system. The (Windows) **PaylinkSerial** utility is available as a part of the released SDK, which takes as a parameter a serial number and programs it into the **only** Paylink unit currently connected to the system.

Operating modes

On Intel platforms and the Raspeberry Pi, the Paylink system can run in one of five different modes. These do not need to be specified explicitly in the configuration file, but are the inevitable consequence of the parameters in the configuration file.

All Paylink configurations require the presence of some Paylink hardware. This hardware may be a full Paylink unit, a Paylink Lite interface or a micro-Paylink dongle.

The five modes that Paylink can run in are:

Standard Paylink: Here the peripherals are controlled by the firmware within the external Paylink unit. This is the “normal” / default mode for the Paylink driver. This is the only mode available on **all** Linux distributions.

Paylink Lite: Here the peripherals are controlled by the PC driver program, using the Paylink Lite 2 interface to access the peripherals. This mode will be used where at least one *Protocol* is specified as “on Paylink Lite”. On Linux this requires the Paylink executable, which is only available for Intel and Raspberry Pi distributions.

Micro Paylink: Here the peripherals are (only) CPI USB peripherals controlled by the PC driver program, with a micro-Paylink dongle providing the authorisation. This mode will be used where a “Using Dongle” line is present in the *System* section. On Linux this requires the Paylink executable, which is only available for Intel and Raspberry Pi distributions.

Merged Paylink: Here the configuration file specifies normal Paylink peripherals and Crane PI USB peripherals as well. The two sets of peripherals are merged together by the driver program, and the result present as a unified whole. This mode is selected where USB peripherals are specified and a “Using Dongle” line is not present. On Linux this requires the Paylink executable, which is only available for Intel and Raspberry Pi distributions.

Merged Lite: Here the configuration file specifies Crane PI USB peripherals and at least one *Protocol* that is specified as “on Paylink Lite”. The two sets of peripherals are merged together by the driver program, and the result present as a unified whole. On Linux this requires the Paylink executable, which is only available for Intel and Raspberry Pi distributions.

Note: If only USB peripherals are specified, but the “Using Dongle” line is not present in the *System* section, then the system will default to expecting a normal external Paylink to be connected - that will continue to support the switches and meter.

External Paylink Peripheral Specification

The Paylink unit contains two types of interface:

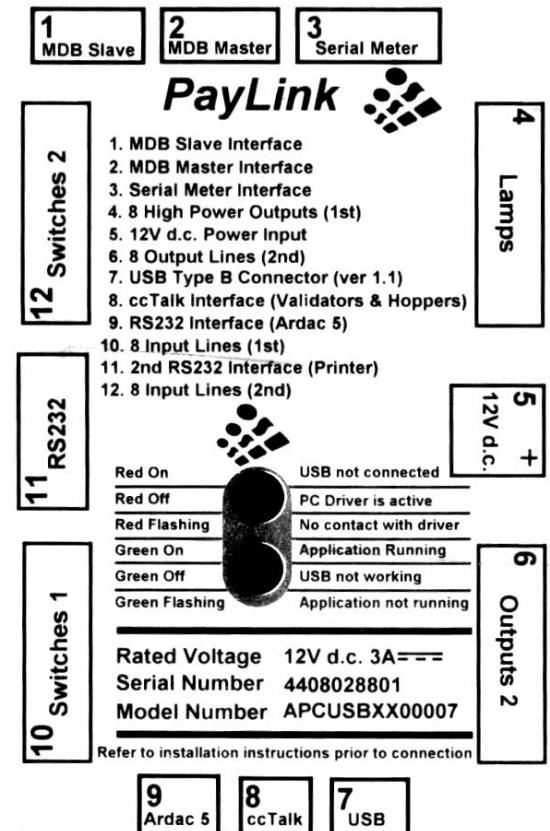
- Interfaces to specific hardware, where the peripheral in question is essentially fully described by the hardware.
- General interfaces to peripherals where one of the Milan / Paylink can be connected to many different peripherals.

The first sort of interface is identified as connectors 3, 4, 6, 10 & 12 on the Paylink lid.

These connectors are provided to connect Milan / Paylink to an SEC meter, switch inputs and LEDs or other outputs.

These interfaces may be connected or not connected, but the item they are connected to is completely defined by the electrical connection (so far as Milan / Paylink is known)

The second sort of interfaces is identified as connectors 1, 8, 9 and 11 on the Paylink lid.



These connectors provide the connections to peripherals concerned with handling currency. In this case there are many possible peripherals that can be connected to the Milan / Paylink unit. The peripherals actually connected are specified in the configuration file.

The Configuration File

The single parameter to Paylink.exe / AESCDriver.exe is a simple text file, containing a set of keywords and values that describe the configuration of the Paylink. Line ends are never significant, and anything on a line after a slash (/) character is ignored.

This section describes the make-up of a configuration file. In the following description:

UPERCASE words represent keywords, which must be spelt as shown, but can be in any case.

[Optional] sections are enclosed in square brackets and represent sections that can be included either to give further details on the item, or to make the “English” read better.

< Values > are enclosed in angle brackets. Values can be number expressed in decimal or hex, or, in some cases, can be pre-defined keywords. A string of digits is held to be decimal number, hex numbers can either start with 0x or end in H.

| Shows that one of the two values on either side if the symbol **must** be included.

There are three top level keywords:

PROTOCOL - Which describes the peripherals connected to the Paylink’s system.
SYSTEM - Which defines the ways in which the overall running of Paylink is modified.
DRIVER - Which gives the driver itself details on how it should run.

Note that to help produce configuration files that are readable, there are a few synonyms and “padding” keywords.

The keywords AT, AND, IS, BY, ON & WITH are all ignored when placed at “obvious” places in the file.

The keywords BILL, BILLS, NOTE & NOTES are all equivalent, and can be interchanged at will.

The keywords MAX, MAXIMUM & FLOAT are all equivalent, and can be interchanged at will.

The keywords CONNECTOR & PORT are equivalent, and can be interchanged at will.

In general plural keywords are equivalent to singular.

DRIVER Details

The **DRIVER** keyword introduces a section of configuration to control the driver program itself:

RUN HIDDEN | VISIBLE

On Windows, this controls the Driver window (independently from the presence of a log file)

- If **HIDDEN** is specified, then no window or taskbar Icon are displayed.
- If **VISIBLE** is specified, then a normal window is shown on the screen.
- If the item is omitted, then a Taskbar Icon appears, but the window is minimised.

On Linux, **RUN VISIBLE** causes the Paylink log to be output to stdout.

USE GLOBAL SEGMENT

On Windows only this specifies that shared memory segment, which is used to communicate with the application, should be in the Global namespace.

This enables IIS to see a normal mode driver, but requires the driver to run with elevated privilege.

LOGFILE “<Name>” [SIZE <K bytes>]

This specifies that a log file is to be generated.

<Name> is the full path to the desired log file, the “ ” symbols are compulsory even if there are no spaces in the name.

<K Bytes> is the maximum size the log file is allowed to reach. If it reaches this size, then it is renamed with a “.old” extension added and a new file started.

If **SIZE** is omitted, the maximum size is 128K bytes.

SERIAL [NUMBER] <Paylink Serial>

This is used in multiple Paylink installations, and specifies the serial number of the Paylink device that *this* driver is to connect to. (This is also available as a Driver program option.)

SYSTEM Details

The **SYSTEM** keyword introduces a section of overall configuration, which includes one or more of:

USING DONGLE

This causes the Paylink driver program to look for a authorization dongle on a system that only use USB connections to devices.

SIMULTANEOUS HOPPERS <Count>

Earlier versions of Paylink would only run a single hopper at a time in order to protect the power supplies. If an installation has the power to run multiple hoppers at once, then this parameter can be used to increase the number of simultaneously running hoppers.

WATCHDOG [On] [OUTPUT] <Pin>

The CheckOperation() facility already allows Paylink to discover that PC application is no longer in contact, and to inhibit all the peripherals. This entry causes output <pin> on the Paylink to be driven only when Paylink is in normal operation, to allow for the control of arbitrary external equipment.

CODE VERSION [BETA | <Number>]

The Windows version of the Paylink driver checks to see if the firmware is a FULL release, and puts up a warning message if not. This entry allows for the suppression of such a message.

POWER ON [ON] [OUTPUT] <Pin> DELAY <MSec>

To allow for the power on sequencing of the elements within a cabinet, this allows the Paylink device to control an output that turns on a specific time after the Paylink powers up. This entry causes output <pin> on the Paylink to be driven <Msec> milliseconds after the Paylink itself powers up.

POWER ON RESET [ON] [OUTPUT] <Pin> DELAY <MSec>

For the situation where a Paylink can be used to correctly reset a PC, this generates a reset pulse of <MSec> at startup, unless the PC driver establishes communication within 5 seconds.

MECHANICAL METER <Meter No> [ON] [OUTPUT] <pin>

This entry can repeated up to 8 times. Each entry defines to the Paylink meter functionality that a mechanical meter <Meter No> is connect to output <Pin>. (More details are above.)

POWER FAIL [ON] [INPUT] <Pin>

This does not appear anywhere in the API. If used, this is a switch connection that should be shorted to ground whenever the power supply is satisfactory and go open circuit as soon as there is a problem. This input is used in two places:

1. As soon as this output triggers, all acceptors are disabled. This primarily allows coin acceptors to reject coins that will not complete correct acceptance before the power fails complete.
2. The mechanical meter processing will not start a pulse if this input is not satisfactory. This removes the possibility of a pulse being cut short by a power failure.

COLOURS <Red D> <Green D> <Blue D> <Red E> <Green E> <Blue E>

This defines the system acceptor colours, as six numbers in the range 0 to 255. The first 3 numbers define a colour to be used for a disabled acceptor, the 2nd three numbers define a colour to be used for an enabled acceptor.

At present this is only implemented on the Innovative SmartPayout.

PROTOCOL Details

PROTOCOL <Name> [ON|FROM] [AUX] [CONNECTOR | PORT | PAYLINK | DLL] <Connector>

This introduces a section describing the usage of the named protocol on the specified connector.

DLL indicates this is a DLL interface, not via a connector

The **Connector** is one of the following:

CCTALK or 8	On Paylink, the six way connector near the USB cable.
MDB or 1	On Paylink, the three way Molex KK at the other end to the USB cable.
RJ45 or ARDAC or 9	On Paylink, RS232 RJ45 connector near the USB cable.
GEN2 or RS232 or 11	On Paylink, RS232 seven way connector.
RS232-2	On Paylink Two, the five way connector.
RS232-3	On Paylink Two, the four way connector.
USB	A direct to peripherals connection for a CPI USB peripheral.
LITE	The peripherals are connected to the PC via a Paylink Lite.
<DLL Name>	The peripherals are connected to the PC via this DLL interface.

Note: The cctalk and MDB protocols are usually used together with unique electrical levels on the connection. Standard Paylink provides these levels on the relevant connectors and so it would be unlikely that these would specify other than the dedicated connector - but doing so is perfectly valid.

If you are using Paylink Lite V2 or Paylink MDB Lite, you specify **LITE**. If you wish to use more than one Paylink Lite unit, the system automatically classifies the additional units as auxiliary in the following ways:

Lite Units in the system	cctalk	MDB	RS232
Base Lite V2 only	x		
Base MDB Lite only		x	
Base Lite V2 RS232 only			x
Base Lite V2 → Aux MDB	x	x	
Base Lite V2 → Aux RS232	x		x
Base MDB Lite → Aux RS232		x	x
Base Lite V2 → Aux RS232 & Aux MDB	x	x	x

The system assumes that you are using a Lite device with a corresponding protocol.

(An Aux RS232 connection is handled by a specially programmed USB / RS232 converter, an Aux MDB is a specially programmed MDB Lite, with no I/O)

The following protocols can be specified. With the exception of **GEN2** and **TFLEX**, each protocol then requires the devices it communicates with to be further specified.

CCTALK	
CCNET	
MDB	
ID003	
EBDS	
MEIBNR	
TFLEX CX25	- CPI TFLEX / CX25 coin dispensers
CLS	- CPI Coin Recycler
F56 F53 F400	- These are synonym of each other for the Fujitsu D-Level protocol.
MFS	
GEN2	- The Future Logic ticket printer protocol.

CCTALK Device Definition

The cctalk protocol handler supports a potentially large number of note / bill acceptor, coin acceptor and payout devices. Following the introductory PROTOCOL entry, **all** the cctalk devices in use on **this** installation have to be defined, as follows:

COIN [ACCEPTOR] [AT] <Address> [BNV <Key>] [CRC]

This specifies that a coin acceptor is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used. Normally coin acceptors are at address 2, and do not use BNV or CRC messages.

BULK [ACCEPTOR] [AT] <Address> [BNV <Key>] [CRC]

This specifies that a bulk coin acceptor is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used. This device will receive special processing as described earlier in the document. Normally bulk coin acceptors are at address 2, and do not use BNV or CRC messages.

SMARTHOPPER [WITH] [ACCEPTOR] [AT] <Address> [BNV <Key>] [CRC]

This specifies that an Innovative smart hopper is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used. Smart hoppers are often at address 7, and probably do not use BNV or CRC messages.

[WITH] [ACCEPTOR]

If the keyword ACCEPTOR is present then this indicated that Paylink should check for and use an acceptor connected locally to the SmartHopper device.

SMARTCOIN [AT] <Address> [BNV <Key>] [CRC]

This specifies that an Innovative smart coin system is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used. These are often at address 7, and probably do not use BNV or CRC messages.

The handler for this is identical to “SmartHopper with Acceptor” except that it will not attempt to pay out denominations with a reported count of zero.

NOTE [ACCEPTOR] [AT] <Address> [No Reset] [BNV <Key>] [CRC]

BILL [ACCEPTOR] [AT] <Address> [No Reset] [BNV <Key>] [CRC]

This specifies that a note / bill acceptor is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used.

Normally note / bill acceptors are at address 40 (28H), and often use a BNV key of 123456 and CRC message validation.

The optional “**No Reset**” keyword supresses multiple resets performed as a part of cctalk automatic note recovery.

<Type> RECYCLER [AT] <Address> [MAX <count> NOTES]

This specifies that a note / bill recycler is to be found at the specified cctalk <Address>, and *may* specify that BNV encryption with the given key and / or that CRC message validation is used. As there is no standard specification for controlling a recycler, the <Type> also has to be specified.

Apart from the MERKUR device which does not use BNV, the device will be a DES type, and hence there is no need to specify a BNV key as that is discovered at the same time as the DES key.

A <Type> from the following list can be used:

MERKUR	- A recycler using the same commands as the Merkur MD100 device.
VEGA	- A recycler using the same commands as the JCM Vega unit
NV11	- A recycler using the same commands as the Innovative NV11
NV200 SMARTPAYOUT	- A recycler using the same commands as the Innovative NV200 / SmartPayout
ICT	- A recycler using the same commands as the ICT BR2300 recycler.

MAX is only valid with Merkur, and specifies a maximum float level of <Count>

HOPPER [AT] <Address> [VALUE <Coin Value>] [AZKOYEN] [READOUT VALUE] [TIMEOUT <Count>] [CRC]

This specifies that a coin hopper is to be found at the specified cctalk <Address>, and *may* specify that CRC message validation is used.

Normally cctalk hoppers are at addresses that start at 3 and increase as more hoppers are added. CPI hoppers can be addresses from 3 to 10.

Hoppers require a significant amount of configuration. Any combination of the keywords in any order can be used.

Note that there are three different ways to specify the coin value; two in the configuration file or the application can set a value that overrides any configured value. Any method can be used, but Paylink will not use a hopper for which a value has not been set. (This may be a desired operational mode.)

[VALUE <Coin Value>]

Specifies the (default) value in pence of the coins in this hopper. If neither of the other two options are used, then this value is fixed.

[READOUT VALUE]

Specifies that the Hopper Eprom will be read during initialisation and any coin value found will be used instead of the default (if any) established by the <Coin Value> entry.

[AZKOYEN]

The detailed implementation of cctalk commands differs between CPI and Azkoyen. This keyword forces Azkoyen specific processing of the hopper. If the hopper replies to the relevant initialisation message with a Manufacturer starting "Azk" then this processing is selected automatically.

[TIMEOUT <Count>]

This specifies that the cctalk header 165 (Modify Variable Set) be used to change the "payout timeout" value from its default of 10 seconds to <Count> periods of 1/3 second. This is unlikely to work on hoppers not made by Crane PI.

Examples

- Hopper at 03 Value 100 - is a hopper whose value is initialised to 100
- Hopper at 03 Readout Value - is a hopper that is unusable unless a coin value can be read out, or set by the application.
- Hopper at 03 - is a hopper that is unusable unless the application sets a coin value.
- Hopper at 03 Value 100 Azkoyen - is a hopper whose value is initialised to 100 and that will unconditionally use the Azkoyen protocol variations.

NOTE ACCEPTOR [IS] ELITE

This must follow a **USB** connector keyword, and specifies that an Elite note is connected directly to the PC via a USB lead.

COIN RECYCLER [IS] CR01x | BCS

This must follow a **USB** connector keyword, and specifies that the corresponding coin recycler system is connected directly to the PC via a USB lead.

COIN RECYCLER [IS] BCR [MAX COINS | FLOAT <level>]

This must follow a **USB** connector keyword, and specifies that a BCR coin recycler system is connected directly to the PC via a USB lead. The optional parameter causes the Paylink start-up code to send the appropriate message to set the level for the hoppers.

CCNet Device Definition

The CCNet protocol allows for addressable devices, although the RS232 electrical connection that is normally used will only allow for a single device to be connected.

RECYCLER [AT] <Address> [SCALE [BY] <Scale Factor>] [EJECT BILL]

At present <Address> must be 1. This specifies that a B2B60, B2B100 or B2B300 bill recycler is connected.

The bill values are obtained from their descriptions, but the recycler has no concept of a denomination scale factor. Paylink assumes a default factor of 100 which is suitable for Euro / Dollar / Pound systems, but other nationalities may specify a factor of 1.

The EJECT BILL keywords are only valid with a B2B60, and cause alternative payout processing to be used, which ejects the bills from the unit as they are paid out, but which prevents monitoring of the progress of the payout.

OPTION BYTES <Byte 1> [<Byte 2> [<Byte 3>]]

This specifies one two or three options bytes as literal numbers to configure the recycler.

SECURITY BYTES <Byte 1> [<Byte 2> [<Byte 3>]]

This specifies one two or three security bytes as literal numbers to configure the recycler.

ESCROW <E Count> [AND RECYCLE <R Count>] BILLS ON CASSETTE <Cassette>

This can only follow a RECYCLER definition and specifies that extended escrow will be used with it.

<Cassette> is the cassette number of the cassette on the acceptor that is to be used for recycling operations. If RECYCLE is not specified, then this cassette will not be visible to the application.

<E Count> is the maximum number of notes that can be held in Escrow, before they have to all be stacked or returned. Escrow uses control storage within Paylink, and so should be set to a sensible number.

<R Count> allows the recycler cassette to also be logically used as a normal recycler cassette. Notes of the correct value that are logically stacked will be physically retained on the cassette and be available for future payout operations.

The total number of notes that will actually fit on a cassette is dependent upon physical constraints, so it is the user's problem to ensure that <E Count> plus <R Count> notes will fit onto a cassette.

ACCEPTOR [AT] <Address> [SCALE [BY] <Scale Factor>]

At present <Address> must be 3. This specifies that a CCNet bill acceptor is connected.

MDB Device Definition

The MDB protocol allows for two changer devices and for two bill acceptors as well as a cashless device. This section states which are used:

CHANGER [AT] <Address>

This specifies that a coin changer is to be found at the specified MDB <Address>. Normally MDB coin changers are at <Address> 08H

NOTE [ACCEPTOR] [AT] <Address>

BILL [ACCEPTOR] [AT] <Address>

This specifies that a note / bill acceptor is to be found at the specified MDB <Address>. Normally MDB note / bill acceptors are at <Address> 30H

CASHLESS [ACCEPTOR] [AT] <Address> [IDLE [AMOUNT] <value>]

This specifies that a cashless (card reader) is to be found at the specified MDB <Address>. Normally MDB cashless devices are at <Address> 10H

The keyword IDLE indicates that level 3 processing should be used if possible. The <Value> field is present because the API had to present a value. The value used here does not affect Paylink processing in any way.

C2 [AT] <Address>

CF7000 [AT] <Address>

CF7xxx [AT] <Address>

These are only allowed following a **USB** connection protocol line and specify that a USB connected coin changer of the quoted type is set up at the specified MDB <Address>, normally 08H.

ID003 Protocol

[WITH] ACCEPTOR | RECYCLER

An ID003 communications line can connect to one single unit. This has to be specified as either a "standard" ID003 note acceptor or to a recycler that is using the JCM UBA recycler ID_003 extensions.

(In the file this will typically follow the <Connector> on the same line.)

EBDS Protocol

[WITH] ACCEPTOR [MAX <Count> NOTES] [SCALE <Scale Factor>]

[WITH] RECYCLER [AND ESCROW] [NO RETURN]

[MAX <Count> NOTES] [SCALE <Scale Factor>]

An EBDS communications line can connect to one single unit. This has to be specified as either a “standard” EBDS note acceptor or as an SCR recycler.

The SCR recycler can run in one of three modes, the normal one is **Single Note Escrow**

The **Multi Denomination Recycling** mode is detected at run time – the interface is similar to the **SNE** one and no configuration file setting is required.

The extended **Multi Note Escrow** is where a number of notes can be escrowed and if necessary those specific ones returned - the optional **AND ESCROW** turns on extended escrow for this processing.

The **NO RETURN** keyword sets the SCR recycler to retain bills in the event that communications with the Paylink is lost.

EBDS bill acceptors can have a very large number of different note identities, which can cause problems with the Paylink firmware. The **MAX <Count> NOTES** overrides the Paylink default of 16. Where the number on the acceptor is greater than the number in Paylink, Paylink folds the identities to overlap each other,

The optional <Scale Factor> is a power of ten that is used to adjust the value read from the acceptor before decoding into the Paylink value.

MEIBNR Protocol

[[WITH] ESCROW]

An MEI BNR bill recycler connected to the Paylink system by the MEI supplied DLL. This line causes the Paylink program to look for and attempt to load a DLL called MEIBNR.DLL. This DLL will only load if the MEI supplied interface DLLs are available.

The option ESCROW parameter indicates that extended escrow will be used with the recycler. No parameters are supplied as the details of extended escrow storage are completely handed by the device.

TFLEX | T-FLEX Protocol

A Tflex coin dispenser can either be connected to a normal RS232 port, or USB.

CX25 Protocol

A CX25 coin dispenser can either be connected to a normal RS232 port, or USB.

CLS Protocol

A CLS Advance Coin Recycler can be connected to USB.

Gen2 Protocol

A GEN2 communications line can connect to one single Future Logic compatible ticket printer.

F56 Protocol

[SPECIAL BILLS] [UK NOTES] DELIVERY [AT] NONE | FRONT | REAR [HOLD ON PROBLEM] [POOL <Count>]

An F56 communications line can connect to one single Fujitsu F56 family bill / note dispenser.

The **SPECIAL BILLS** keywords set the polymer note configuration flag.
The **UK NOTES** keywords set the configuration flag for the new UK £5.

Paylink *requires* the specification of the note delivery system, so as to know whether to issue a delivery command, and which one.

On an F56, the **HOLD ON PROBLEM** option sets a processing flag so that Paylink will only deliver bills when there has been no problem in the payout.

On an F56, the **POOL** option sets the number bills that are collected at the pool stage before being delivered by Paylink. The default if this is not specified is 50 bills.

MFS Protocol

An MFS communications line can connect to one single MFS bill / note dispenser.

Cassettes

The cassettes for bill Dispensers can optionally be configured.

F53 / F56 / F400 cassettes are identified, usually by one or two magnets fitted into the case that identify the bills / notes that are loaded into the *cassette* (**not** the position in the machine). This section maps these identities onto specific notes.

CASSETTE [WITH] <Identity> VALUE <Value> [MAX <Max>] [MIN <Min>] [THICKNESS] <thickness>

This specifies that the cassette with the given magnet <Identity> contains bills / notes with the given value (in cents / pence).

Paylink will default to allowing any size note to be accepted, but for added security correctly encoded length and thickness bytes <Max>, <Min> and <thickness > *can* optionally be specified and will be sent to the F53/F56 during cassette initialisation.

Examples

F56 cassettes typically contain two magnets, and so a common standard configuration uses the six two bit numbers:

```
Cassette 3 Value 10000
Cassette 5 Value 5000
Cassette 6 Value 2000
Cassette 9 Value 1000
Cassette 10 Value 500
Cassette 12 Value 100
```

Less common is one magnet, but an alternative set uses the four one bit numbers:

```
Cassette 1 Value 10000
Cassette 2 Value 1000
Cassette 4 Value 500
Cassette 8 Value 100
```

Original Paylink Definition.

The definitions required to reproduce the original Paylink configuration are:

```
Protocol cctalk on connector cctalk
  Coin Acceptor at 2
  Note Acceptor at 40 BNV 123456 CRC
  Hopper at 3 Value 100 Readout Value
  Hopper at 4 Value 40 Readout Value
  Hopper at 5 Value 25 Readout Value
  Hopper at 6 Value 20 Readout Value
  Hopper at 7 Value 10 Readout Value
  Hopper at 8 Value 5 Readout Value
  Hopper at 9 Value 200 Readout Value
  Hopper at 10 Value 1 Readout Value
```

```
Protocol ID003 on connector RJ45
```

```
Protocol GEN2 on connector 11
```

```
Protocol MDB on connector MDB
  Changer at 08H
  Bill at 30H
```

Disclaimer

This manual is intended only to assist the reader in the use of this product and therefore Aardvark Embedded Solutions shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any incorrect use of the product. Aardvark Embedded Solutions reserve the right to change product specifications on any item without prior notice