Milan / Paylink System Manual

Table of Contents

Table of Contents **2**
Revision History **5**
Introduction **6**
Purpose of Document 6
Intended Audience 6
Associated Document(s) 6
Naming 6
Supported Facilities 7
Version Numbering 7
Document Structure 7
Paylink **8**
Installation 8
Money representation 8
Acceptance 8
Payment 8
Payout Function *8*
PaySpecific Function (1.12.6) *9*
End of payout processing. *9*
Auxiliary Items 10
Switch inputs *10*
Outputs *10*
Meters *10*
System Structure 10
USB Connection *11*
Troubleshooting 11
Coin / Note Acceptor Usage Details **12**
Token Handling (Coin Ids) (1.11.x) 12
Dual Currency Handling (Coin Ids) (1.11.x) 12
Coin Routing. 13
Route coins to a general cash box *13*
Route specific coins to a specific cash box. *13*
Route coins to a hopper until it is full then route it to a coin cash box. *13*
Paylink Routing - Flow Diagram *14*
Control of Motorised Acceptors 15
ccTalk bulk coin acceptor (1.11.3) *15*
BCR / CR10x coin recyclers (1.11.5) *15*
MDB changer / BCR / CR10x recycler support. 16
MDB Payout *16*
MDB tube level monitoring *17*
Read out of Acceptor Details (1.11.x) 18
Coin / Note Dispenser Usage Details **19**
Dispenser Power Fail support. 19
Detailed Device Support 19
Abandoning a payout in progress (1.11.3) *19*
Control of unwanted bill payout (1.11.3) *19*
Combi Hopper Support 19
Read out of Dispenser Details (1.11.x) 20

Bill Recycler Operation (1.12.3) 21
Introduction 21
Security 21
DES Key Exchange 21
Component Identity 22
Routing 22
Dispenser Destination 22
Routing Control. 22
Dispenser Emptying 23
Full Dump 23
Partial Dump 23
Payout Progress 23
Cancelling Payout 23
Notification of progress 23
Power Fail 24
Temporary power interruption 24
Full power Failure 24
Unpaid Bills 25
Device Specific Functionality 25
Cashcode B2B-300 25
Cashcode B2B-60 25
Merkur 100 25
Innovative NV11 Recycler (DES) 25
Innovative NV200 Recycler (DES) 26
JCM UBA Recycler 26
JCM Vega (DES) & JCM UBA Recycler 26
F56 / F53 Bill Dispenser 26
Extended Escrow (1.12.6) 28
Introduction 28
Functionality 28
Accepting Notes 29
Returning Notes 29
Keeping Notes 29
Operation 30
Abnormal Situations 31
Meters / Counters 32
Mechanical Meters (1.12.4) 32
Events (Faults / Auditing) 33
Introduction 33
cctalk coin processing 34
Fault Events 34
Coin Events 34
cctalk note processing 35
Fault Events 35
Note Events 35
cctalk hopper processing 36
ID-003 note processing 38
Fault Events 38
CCNet note processing 39
Fault Events 39
Firmware reprogramming 40
Command Line Options 41

Limitations 41
Milan / Paylink Driver Program Configuration 42
Driver Parameters 42
Multiple Paylink Unit Support. 43
Unit Identification 43
Operating modes 43
External Paylink Peripheral Specification 44
The Configuration File 45
DRIVER Details 46
SYSTEM Details 47
PROTOCOL Details 48
CCTALK Device Definition 48
CCNet Device Definition 51
MDB Device Definition 51
ID003 Protocol 52
F56 Protocol 52
Original Paylink Definition. 53
Disclaimer 54

Revision History

Version	Date	Author	Description
1.0	25th 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

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 around a Paylink unit. That document covers the configuration setting that are used to describe the units connected to Paylink, and the way in which such units are controlled.

The companion document “Milan / Paylink Application Program Interface Manual” is written 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 made and sold by Money Control under licence from AES. There are at present three versions of Paylink: Paylink — The original, metal cased version. Paylink Lite — A much smaller, plastic cased version with a reduced function set. uPaylink — (Micro Paylink) a PC software only version, for use with Money controls USB peripherals.
PCI Card	This is the original obsolescent hardware unit. It was known as Milan a long time ago, but this is its current name,

Supported Facilities

It should be noted that this document cover all versions of the Milan / Paylink system, even those versions that are not yet generally available.

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 hay 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 release 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 is 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.

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 request 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 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.

End of payout processing.

As money is paid out during either process, Paylink updates the total of all credit paid out.

Whilst either of the payment commands are being processed, Paylink reports the status of the system as on-going - when it stops it will either report successful (indicating the value request has been paid out) or it will report the last failure code that was processed. The amount actually paid out can be obtained by comparing the current lifetime credit paid out value with that immediately before the pay out started.

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.

Auxiliary Items

Switch inputs

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 of the number of close, and open of 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.

Outputs

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 Paylink board provides an easily accessible source to enable LEDs to be easily connected to these output.

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 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 lost of a single pulse.

System Structure

A system that uses a Milan / Paylink unit comprises a users application that communicates via a DLL to the Milan firmware installed on the Milan / Paylink unit.

As the Milan / Paylink unit is connected to the PC by a USB cable, there has to be software at both ends of this USB link to conduct the communications over the link.

At the Milan / Paylink end, this communications software is just built into the firmware. At the PC end, the communications software runs as a separate Windows item.

USB Connection

USB Lead

The 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, may cause the 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 Paylink reset, or for 10 seconds after a USB driver program (re-)start.

USB Driver Program.

As detailed in the "Milan / Paylink System Manual", the supplied USB driver program Paylink or 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 causes exception processing to be undertaken as above, and is not recommended.

In the theoretical model of the Milan / Paylink system there were originally intended to be two options for this, a normal windows service, installed and managed in the same way as for other services and as an alternative a Windows program, AESWDriver.exe.

In practice the service option was never completed, and all Milan / Paylink installations prior to release 1.12.1 / 25.12.1 have used the AESWDriver.exe.

This original program has now been superseded by a new configuring driver program which is available in two distinct forms: Paylink.exe which only runs under Windows where it also supports Paylink Lite 2 and USB peripherals made by MCL and AES**C**Driver.exe, the generic driver program, primarily used on Linux.

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.

Troubleshooting

As detailed below in the configuration section, the Paylink driver program provides for 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.

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 first characters of the name of the coin (**`Coin.CoinName`**).

Note: The exact values returned are dependent upon the acceptor manufacturers and hence can not 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.

Coin Routing.

Paylink provides facilities to partially automate the routing of coins to fill a coin dispenser and one or more cash boxes. 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.**

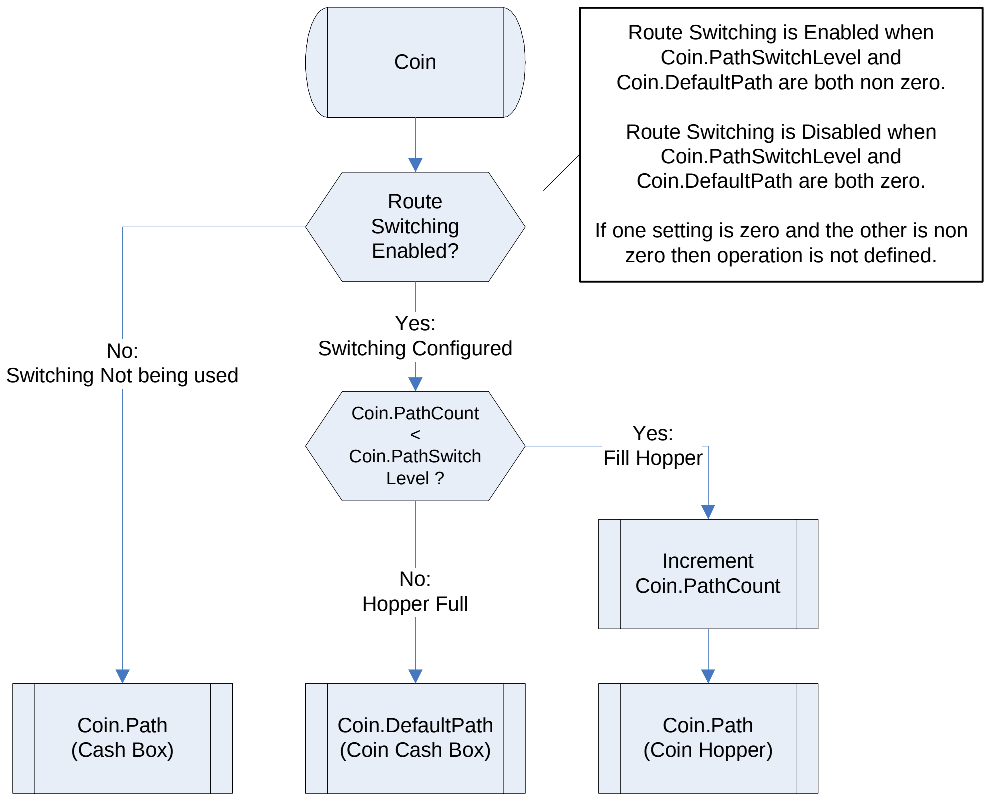
e.g. General Cash box on route 4, coin 1 goes to a hopper on route 1 and a cash box on route 2. Coin.PathCount is 100 and there is space for 300 more coins in the hopper.

• 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


Coin
Route Switching Enabled?
Route Switching is Enabled when Coin.PathSwitchLevel and Coin.DefaultPath are both non zero.
Route Switching is Disabled when Coin.PathSwitchLevel and Coin.DefaultPath are both zero.
If one setting is zero and the other is non zero then operation is not defined.
No: Switching Not being used
Yes: Switching Configured
Coin.PathCount < Coin.PathSwitch Level ?
Yes: Fill Hopper
Increment Coin.PathCount
No: Hopper Full
Coin.Path (Cash Box)
Coin.DefaultPath (Coin Cash Box)
Coin.Path (Coin Hopper)

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 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 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 one, which is supported on most level 3 changers. 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, if the extended mechanism is present, to provide an additional dispenser which is run using the extended mechanism. Where an extended mechanism dispenser is available, the individual tubes are pre-set to inhibited.

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

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.

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

ccTalk	The replies to: • 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.
MDB	N/A (Integral part of Acceptor)
F56	The 12 character firmware revision, followed a '~' followed by the 32 character device information.

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

ccTalk	The binary reply to the ID Serial No (242) command.
MDB	N/A (Integral part of Acceptor)
F56	Not Available

Bill Recycler Operation (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 has meant that the Paylink model has had to enhanced to include these. The approach adopted has been 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.

This section describes the evolved handling of recyclers (and bill dispensers) that is in place in versions 1.12.3 and later.

Security

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

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.
cctalk	is a very versatile system, which can be used in high value applications. The connection is relatively vulnerable to interference and so encryption is used to provide security. The latest security option for cctalk is **DES** encryption, which involves the exchange of local, random keys.

Paylink has, as a philosophy, the idea that it will connect to anything. Using **DES** encryption for a bill and a coin acceptor do not therefore make sense and people wishing to create a fully secure system using these are referred to the DES Paylink system (which *is* closed down and secure)

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 all new cctalk recyclers supported must use **DES** encryption.

As a special case the (expensive) Merkur ccTalk recycler is supported - this is not a general purpose protocol and Merkur recyclers are not expected to be used in vulnerable systems.

DES Key Exchange

Before using a 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 these devices is that the acceptor part usually contains the overall description of the unit, and the dispensers are (relatively arbitrarily) identified by a sequence number (from 1 upwards) and the value that they dispense.

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 a "sequence" number that identifies the dispenser. If the unit only has one dispenser, the field will contain 1.

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

Routing

Dispenser Destination

Paylink, during its initialisation of the unit, determines the value of the coin / bill in the dispenser(s) and which coins / bills are routed into which dispensers. The "sequence" number(s) are stored into the `AcceptorCoin.Path` field(s). All the other `AcceptorCoin.Path` fields will be zero.

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

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 and 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 some bill recycler units, such as the Merkur MD100 and MDB Changers, the routing is fixed and it is not possible for Paylink, and hence the application, to change this.

For other units, the routing can be changed by Paylink. The application notifies Paylink of the desired routing by changing the Path fields of the incoming Coin (Bill) array item to contain the associated dispenser serial number.

Options available are:

- If the Path field for a currently recycled bill is set to zero, the unit will 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 bills are currently stored they may remain available to be paid out (depending upon the device capabilities)
- The application should not set two or more separate bills to contain the same value in the Path field, f this is done, then this situation is undefined
- For all coin / bills with a non-zero path number, if the path field for a bill corresponds to the sequence number for a dispenser, then Paylink will update the recycler to direct that bill into the corresponding dispenser. As a part of this process, many units will cause any bills already in the dispenser will be stored into the cashbox.

Note that as the specification is "where to send the bill" it is not possible to have two dispensers regarded as having the same value bill.

The options for automatic re-routing using DefaultPath and PathSwitchLevel are only available with coin acceptors. For note recyclers, these fields are not used.

Dispenser 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

Cancelling 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, so in the usual case all the coin dispensers should be disabled at the same time as the bill recycler.

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

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.

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 dispense, 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 dispense 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 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:

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.

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.

Innovative NV11 Recycler (DES)

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

Innovative NV200 Recycler (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.

JCM UBA Recycler

This unit can have bills “manually” loaded into the storage stackers. When this occurs, the UBA 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 UBA 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.

When the counter of the number of bills in a 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 stacker runs out of bills when the UBA counter is non-zero, the UBA will zeroize the counter.

JCM Vega (DES) & JCM UBA Recycler

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 is command also used in those recovery situations where the application has not been informed that the bill has been dispensed.

This can help avoid the situation where payouts that were uncompleted actually occur.

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.

These F56 have 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 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.

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 returned to the user. To allow the application control over the timing of this return, the extended escrow system transitions from `OFF` to `RETURNED_PROBLEM` when the `START` command is issued.

Meters / Counters

The 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 cateogizations. 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 refered 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.

The relevant functions calls and structures for these events follow - details on the make up of the event codes are given in the “Milan / Paylink System 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 the "self Test" are ignored - the dispense coins command is dent to the hopper regardless.

On a regular basis the "Test Hopper" command is sent to the each hopper 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 between 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.

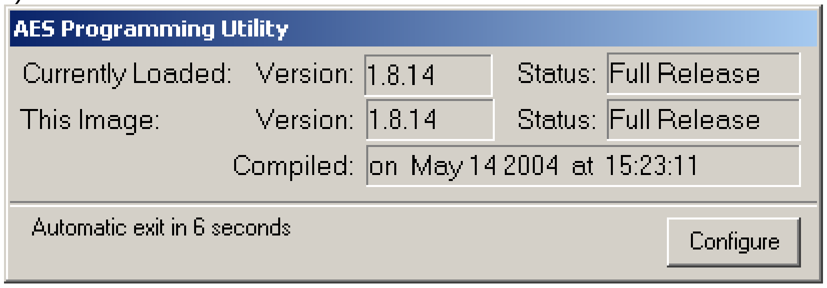
Status Value	Name	Event Classification	Raw Event
0x1C 0x68	REJECTING	INHIBITED	0x68
0x1C nn	REJECTING	REJECTED	nn
0x41	DROP CASSETTE FULL	OUTPUT_PROBLEM	0x41
0x42	DROP CASSETTE REMOVED	OUTPUT_PROBLEM	0x42
0x43	JAM IN ACCEPTOR	MISREAD	0x43
0x44	JAM IN STACKER	OUTPUT_PROBLEM	0x44
0x45	CHEATED	FRAUD	0x45
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		

Firmware reprogramming

All Milan firmware releases are distributed as self-extracting Windows executables. As well as those stored on the distribution CD, they are also accessible 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 Milan / 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.)


AES Programming Utility

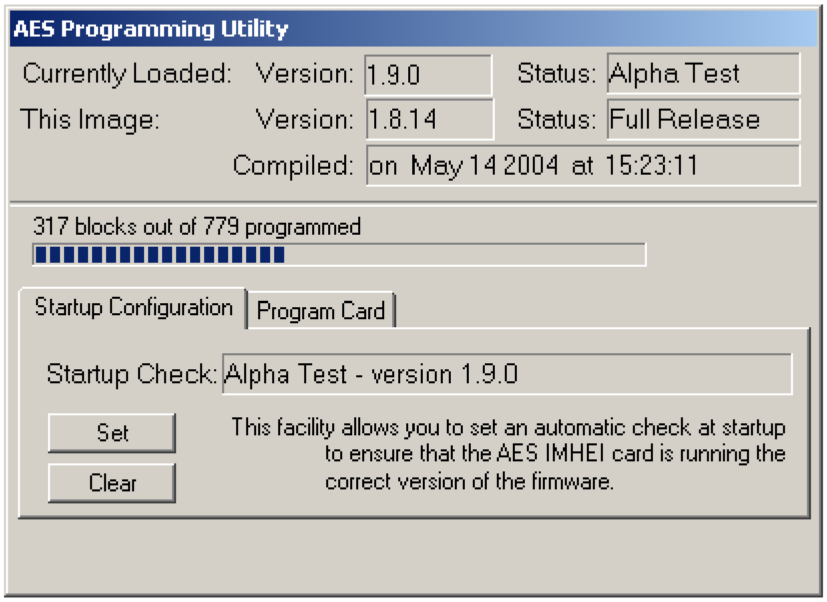
Currently Loaded: Version: 1.8.14 Status: Full Release

This Image: Version: 1.8.14 Status: Full Release

Compiled: on May 14 2004 at 15:23:11

Automatic exit in 6 seconds

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


AES Programming Utility

Currently Loaded: Version: 1.9.0 Status: Alpha Test

This Image: Version: 1.8.14 Status: Full Release

Compiled: on May 14 2004 at 15:23:11

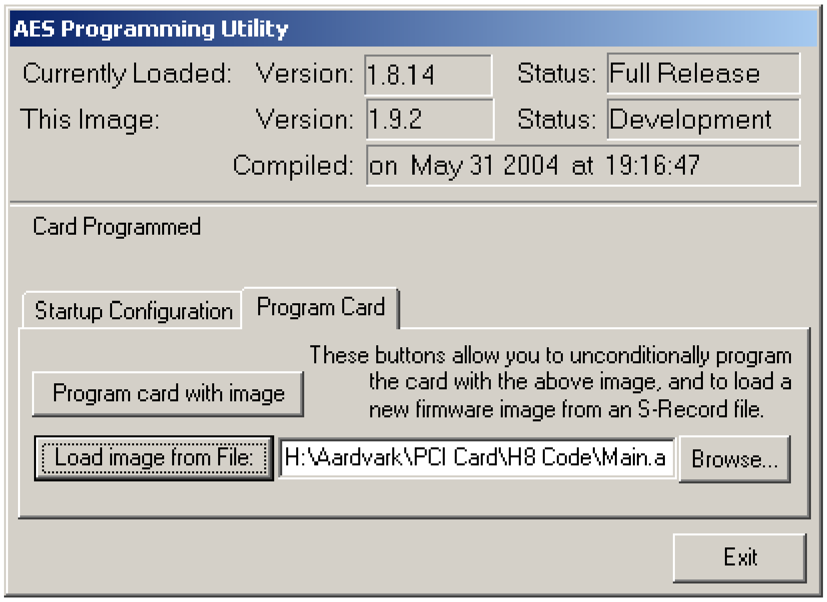
317 blocks out of 779 programmed

Startup Check: Alpha Test - version 1.9.0

This facility allows you to set an automatic check at startup to ensure that the AES IMHEI card is running the correct version of the firmware.

“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.



AES Programming Utility

Currently Loaded: Version: 1.8.14 Status: Full Release

This Image: Version: 1.9.2 Status: Development

Compiled: on May 31 2004 at 19:16:47

Card Programmed

These buttons allow you to unconditionally program the card with the above image, and to load a new firmware image from an S-Record file.

Load image from File: H:\Aardvark\PCI Card\H8 Code\Main.a

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, the requires the use of a special serial cable, the Hitachi programming utility FlashSimple and an S-Record (.a37) file.

For full details on using these, please contact Money Controls.

Milan / Paylink Driver Program Configuration

When initially released only a limited number of peripherals could be connected to Paylink and three (later 4) variants of the firmware were produced, one for each peripheral set.

Since then, the number of possible configurations has grown to the point where some method is needed to tell Milan / Paylink details about where to look for specific peripherals, and how it should regard those peripherals when it found them.

An essential component of the Paylink system is the Diver program, and the method chosen to specify the configuration of the system is to describe the peripheral configuration in a simple text file, and to download that file to the Milan / Paylink unit as the system starts up.

When contact is established between Paylink and the Driver program, 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.

If a system design so requires it, the configuration can be stored by running the driver program during commissioning and the not specifying a configuration to the live driver. This is *not* expected to be a common situation.

Driver Parameters

When Paylink.exe / AES**C**Driver.exe is run it needs to find the 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.

The exact 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 Paylink 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 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 **AESWDriver** 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 Windows (only) 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 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:

Normal 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 Linux. Linux does not support a Paylink Lite interface or a micro-Paylink dongle.
Paylink Lite:	Here the peripherals are controlled by the PC driver program, using the Paylink Lite interface to access the peripherals. This mode will be used where a *Protocol* is specified as "on Paylink Lite".
Micro Paylink:	Here the peripherals are Money controls 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.

Merged Paylink: Here the configuration file specifies normal Paylink peripherals and Money control USB peripherals. 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.

Merged Lite: Here the configuration file specifies Money control USB peripherals and a *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.

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 - which will continue to support the switches and meter.

External Paylink Peripheral Specification

The Paylink unit contains two types of interface:

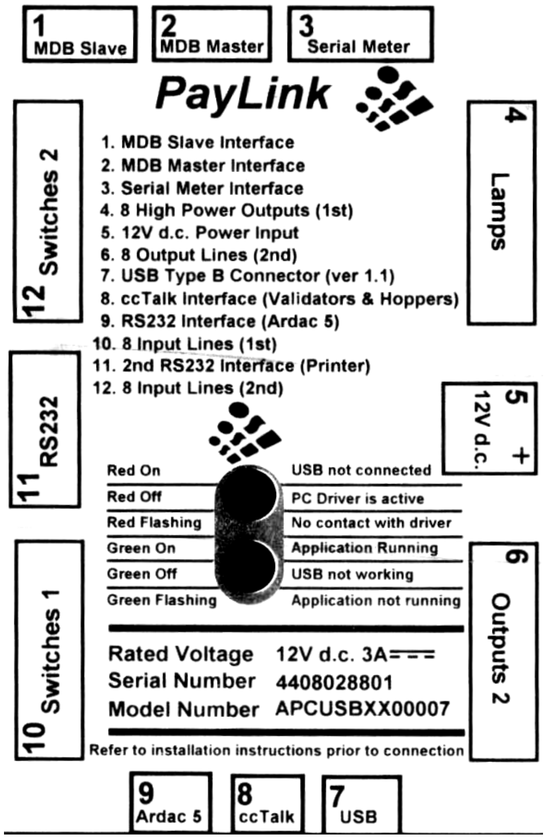
- Interfaces to specific hardware, where the peripheral in question is essentially fully described by the hardware.
- General interfaces to peripherals where one of the Milan / Paylink can be connected to many different peripherals.

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 that map onto numbers. 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, 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.

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 VISBLE 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.

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:

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.

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.

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 is the range 0 to 255. The first 3 numbers define a colour to used for a disabled acceptor, the 2nd three numbers define a colour to used for an enabled acceptor.

At present this is only implemented on the Innovative SmartPayout.

PROTOCOL Details

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

This introduces a section describing the usage of the named protocol on the specified connector. The **Connector** is 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 Molex KK connector on the long side.
USB	A direct to peripherals connection for a Money Controls USB peripheral.
LITE	The peripherals are connected to the PC via a Paylink Lite.
[AUX]	is only valid with **LITE** and indicates that this is a 2nd Paylink Lite unit, with no I/O.

Note: The cctalk and MDB protocols are usually used together with unique electrical levels on the connection. 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.

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

CCTALK	
CCNET	
MDB	
ID003	
F56 | F53 | F400	- These are synonym of each other for the Fujitsu D-Level protocol.
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 ccctalk 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.

NOTE [ACCEPTOR] [AT] <Address> [BNV <Key>] [CRC]
BILL [ACCEPTOR] [AT] <Address> [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.

<Type> RECYCLER [AT] <Address>

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 no 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

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 and use an acceptor connected locally to the SmartHopper device.

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

This specifies that a coin 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.

Normally cctalk hoppers are at addresses that start at 3 and increase as more hoppers are added. MCL 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 following 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 Money Controls 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 Money Controls.

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 be unconditionally use the Azkoyen protocol variations.

NOTE ACCEPTOR [IS] ELITE

This must follow a USB 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 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 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.

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 dependant 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. 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

ID003 Protocol

[WITH] ACCEPTOR | RECYCLER

An ID003 communications line can connect to one singe 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.)

F56 Protocol

DELIVERY [AT] NONE | FRONT | REAR

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

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

The cassettes for this unit 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 form 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