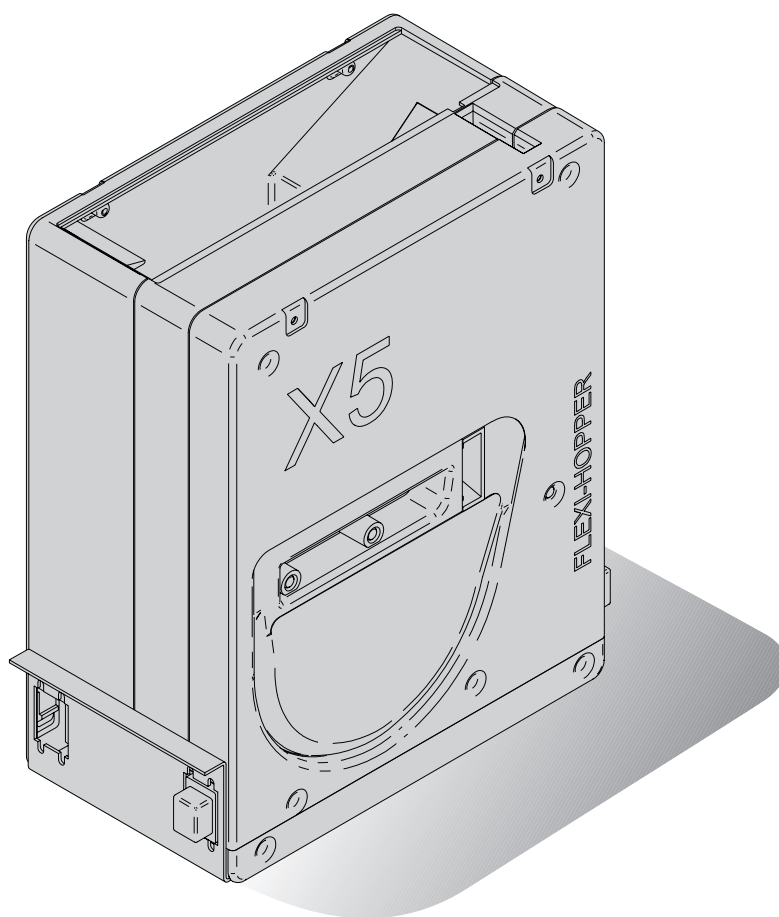


**MICROHARD**  
VENDING PROJECTS

**COIN AND/OR TOKEN PAY-OUT**

# **X5 Family**

## **ccTalk protocol/AES**



Document reserved according to the law, with reproduction or transmission to third parties is strictly prohibited without the explicit authorization of the company **MICROHARD s.r.l.**

Components may be updated and therefore present different details from those depicted below, without this constituting a detriment due to the texts contained in these instructions.

**MICROHARD s.r.l.** is not responsible for accidents, breakage, etc. owing to the persons being unaware or not applying the guidelines contained in these instructions. The same applies to changes and variations and/or use of unauthorized parts.

## **INDEX**

### **1 GENERAL INFORMATION**

- 1.1 DESCRIPTION
- 1.2 MAIN FEATURES
  - 1.2.1 COIN OUTPUT
  - 1.2.2 ERROR CODE
  - 1.2.3 PCB POSITION
  - 1.2.4 ccTALK STANDARD
  - 1.2.5 CONNECTOR POSITIONS
  - 1.2.6 ANTI-JAM SYSTEM
- 1.3 SAFETY
- 1.4 DIMENSIONS
- 1.5 TECHNICAL DATA
- 1.6 POWER SUPPLY FOR MOTOR

### **2 INSTALLATION**

### **3 ELECTRICAL INFORMATION**

- 3.1 GENERAL DESCRIPTION
- 3.2 POWER SUPPLY
- 3.3 OPERATING MODES
- 3.4 OPTICAL SENSORS
- 3.5 LED INDICATORS
- 3.6 COIN LEVEL PLATES

### **4 ELECTRONIC SPECIFICATIONS**

### **5 SPARE PARTS**

## 1 GENERAL INFORMATION

### 1.1 DESCRIPTION

**X5** is a new coin and token distributor and can be used in various applications such as payment kiosks, automatic cash registers, slot machines and money changing and coin recycler machines

### 1.2 MAIN FEATURES

5 product configurations and operating modes

#### 1.2.1 COIN OUTPUT

By using various accessories provided with **X5** you can choose from 5 different coin or token output modes by diverting the flow from the pre-set position.

#### 1.2.2 ERROR CODE

When the yellow LED on the **X5** turns on it means the following: that the device has power; the error code is indicated by means of a series of different flashes, enabling rapid identification of the causes of the malfunction or counting the coins paid according to ordinary operation.

#### 1.2.3 PCB POSITION

The PCB enables all the **X5** functions -**Pay-out** can be updated from the outside without have to dismantle the distributor's parts.

#### 1.2.4 ccTALK STANDARD

For the **X5 Pay-out** versions the unit functions by using the ccTalk standard. You can choose the device with the Cinch 12 connector or with the ccTalk standard 10 pins cup connector. The parallel version is available with the cinch connector only.

#### 1.2.5 CONNECTOR POSITIONS

The connector for ccTalk operation or the 12 pin (Cinch) connector can be set in two different positions:

- side opposite coin output window
- side of coin output window (reverse)

#### 1.2.6 ANTI-JAM SYSTEM

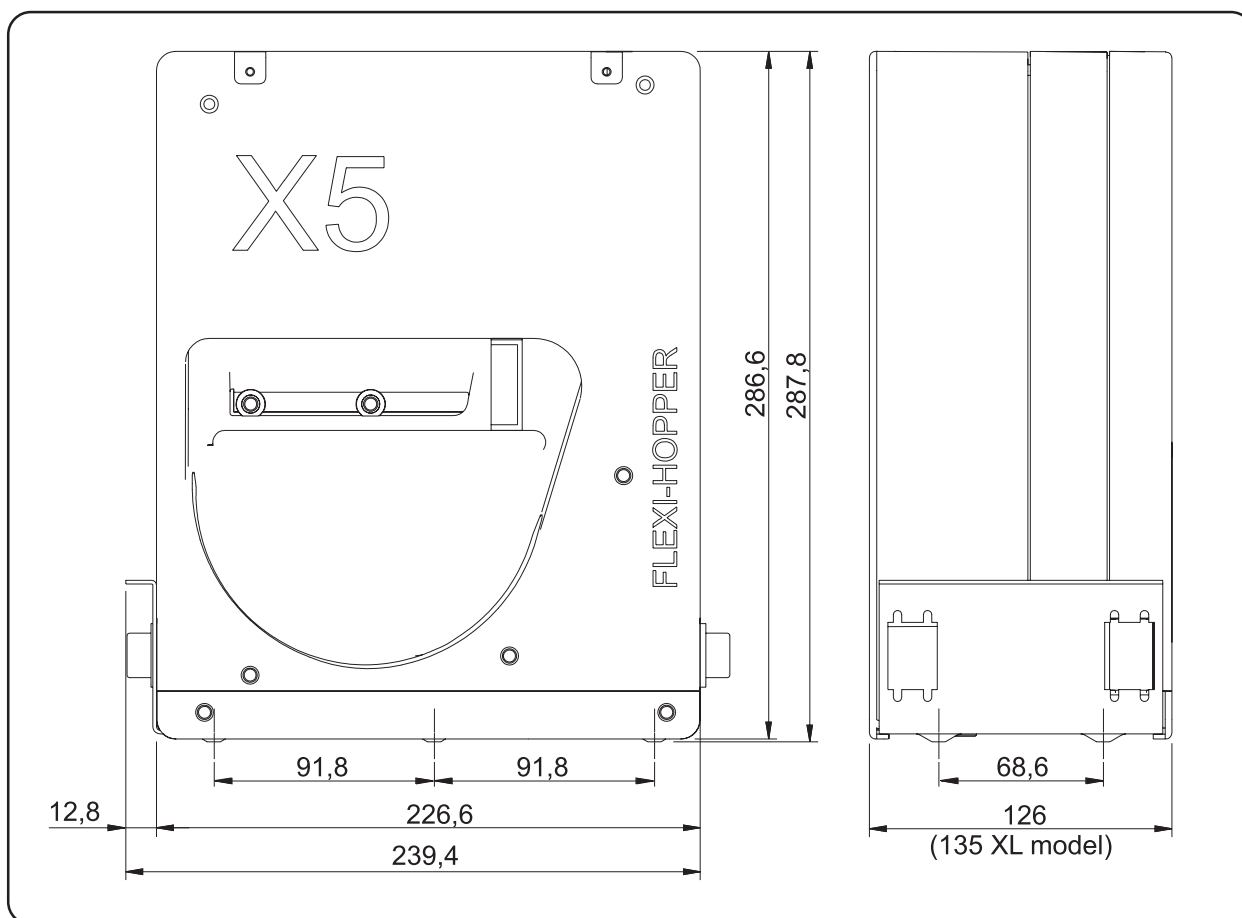
If the motor or belt jams the motor stops and subsequently restarts in reverse; it then stops again and restarts in the correct direction. If this does not occur the operation will be repeated with 3 tries.

### 1.3 SAFETY



**X5** should not be connected/disconnected from the slide base with the power supply on.  
Do not insert hands into the X5 while operating as there are moving mechanical parts.

## 1.4 DIMENSIONS



## 1.5 TECHNICAL DATA

Distribution speed	4 coins/sec
Coin capacity	1500 pieces of 1.00 •
Weight (empty)	2 kg
Diameter coins dispensed	from 16 to 39 mm (from 20 to 31.5 mm with standard chain)
Thickness coins dispensed	from 1.25 to 4.5 mm

## 1.6 POWER SUPPLY

	<i>standby</i>	<i>mpty</i>	<i>max load</i>	<i>forced stop</i>
MOTOR 24Vdc $\pm$ 10%0	mA	80 mA	500 mA	(transient) 500 mA
LOGIC 12Vdc $\pm$ 10%8	0 mA	80 mA	80 mA	-

**standby:** X5 stopped but with power on

**empty:** normal operation

**max load:** operation with coin hopper full

**forced stop:** refers to value of current absorbed by motor over which it is jammed, and the anti-jam procedure begins.

## 2I INSTALLATION



**DO NOT SUPPLY X5 WITH POWER UNTIL ASSEMBLY AND DUE INSPECTION ARE COMPLETED**

- A ffix the **X5**'s slide to the machine.
- Check that it is properly plugged in if using 12 pin standard connector.
- Hook up the flat or 12 pin connector, following the instructions for each single pin as shown in paragraph 4.1, using a suitable cable to support currents and maximum voltages.
- I nsert **X5** onto the slide until it is totally inserted.
- T urn on electric power.

## 3E ELECTRICAL INFORMATION

### 3.1 GENERAL DESCRIPTION

The operating modes of **X5** are guided by a microprocessor:

ccTalk/AES protocol  
parallel protocol  
multi coin protocol  
coin counter and divider

### 3.2 POWER SUPPLY

**X5** is equipped with a 24 v continuous feed motor.

### 3.3 OPERATING MODES

#### **X5-cc-Talk/AES MOD.**

Functions with ccTALK/AES protocol.

### 3.4 OPTICAL SENSORS

There is a pair of optical sensors to determine the coins paid (including coins with a central hole) and a pair of inductive sensors for the multi-coin models.

#### **X5-ccTalk/AES MOD.**

By means of the data line, ccTalk/AES protocol monitors all the sensors' functions.

### 3.5 LED INDICATORS

#### **X5-ccTalk MOD.**

This has just one flashing green LED:

rapid flashing indicates that the **X5** is distributing one coin per flash

flashes with longer intervals indicates that the **X5** is in on

led constantly on means **X5** is in error: this can include photocell error or corrupt data in EEPROM or insufficient power supply.

### 3.6 COIN LEVEL PLATES

Inside **X5** there are brass plates to determine the level of coins.

#### **X5-cc-Talk/AES MOD.**

Plate signals are internally operated by the ccTalk/AES protocol.

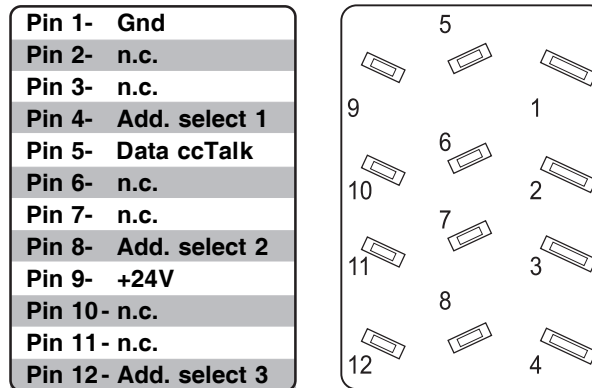
## 4- ELECTRICAL SPECIFICATIONS



**N.B. TO CONNECT THE X5 USE A 22AWG CABLE.**

### 4.1 X5-cc-Talk/AES MOD. CONNECTOR

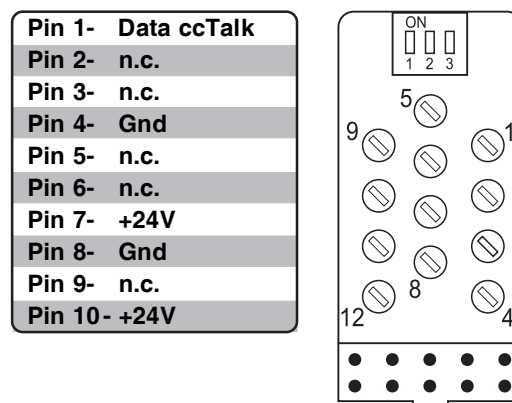
12 Pin connector (12V ccTalk, 24V ccTalk versions)



Pin 1 and Pin 9                      power supply  
 Pin 4, Pin 8 and Pin 12        X5 address (as described in paragraph 4.2)  
 Pin 5                                  data line

The remaining Pins are not connected.

Optional 10-Pin connector mounted on optional MH455a board (versions 12V ccTalk, 24V ccTalk)

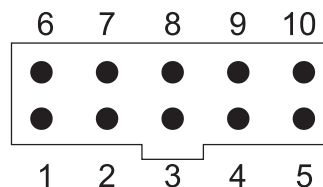


Pin 1                                  data line  
 Pin 4 and Pin 8                  GND  
 Pin 7 and Pin 10                +24

The remaining Pins are not connected.

## Flat 10 Pin Connector (12V ccTalk, 24V ccTalk versions)

Pin 1-	Data ccTalk
Pin 2-	n.c.
Pin 3-	n.c.
Pin 4-	Gnd
Pin 5-	n.c.
Pin 6-	n.c.
Pin 7-	+24V
Pin 8-	Gnd
Pin 9-	n.c.
Pin 10-	+24V

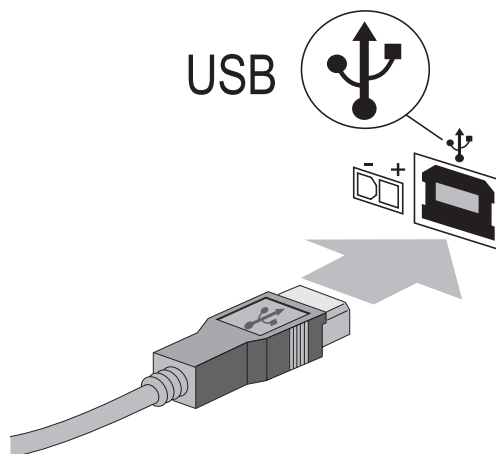


**Pin 1** data line  
**Pin 4 and Pin 8** GND  
**Pin 7 and Pin 10** +24

The remaining Pins are not connected.

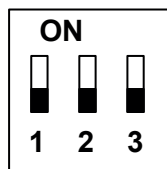
## USB Connector 24V ccTalk version)

Separate power supply 24VDC - 12VDC (optional)  
 on connector "mini-fit" 2 poles.



## DPO START LIST CONTROLS

The serial address of the hopper can be set using the Dip-Switch group.  
 The address changes depending on the combination of Dip-Switch, following the table:



Dip-Switch 1	Dip-Switch 2	Dip-Switch 3	N° hopper	serial address
			1	3
		X	2	4
	X		3	5
	X	X	4	6
X			5	7
X		X	6	8
X	X		7	9
X	X	X	8	10



**Header 254 “Simple poll”**

Sent byte: none

Received byte: none

The hopper replies ACK if the command is identified or the address is correct

**Header 253 “Address poll”**

Sent byte: none

Received byte: only 1 byte (non-standard format) alike the using address and sent with a delay of  $4 \times \text{address ms}$ .

**Header 246 “Request manufacturer ID”**

Sent byte: none

Received byte: String that indicates the builder. In this case “MicroHard.Srl.”

**Header 245 “Request equipment category ID”**

Sent byte: none

Received byte: String that indicates the kind of peripheral device. In this case “Coin Inject System”

**Header 244 “Request product code”**

Sent byte: none

Received byte: string that indicates the kind of product code. In this case “X5 DPO”

**Header 242 “Request serial number”**

Sent byte: none

Received byte: 3 byte [byte1],[byte2],[byte3] that indicates the serial number of the peripheral device. The first byte is always LSB so the serial number is  $[\text{byte1}] + 256 \times [\text{byte2}] + 65536 \times [\text{byte3}]$

**Header 241 “Request software revision”**

Sent byte: none

Received byte: string with soft and hardware versions. Ex. “1.0 1.0” indicates software rev 1.0 and hardware rev 1.0

**Header 217 “Request Payout Hi-Lo status”**

Sent byte: none

Received byte: 1 byte that indicate the status and the presence of sensors of minimum and maximum

Bit0=1 Coins do NOT overtake the minimum sensor (if=0 they overtake)

Bit1=1 Coins do NOT overtake the maximum sensor (if=0 they overtake)

Bit2=1 Minimum sensor present (if=0 absent)

Bit3=1 Maximum sensor present (if= absent )

**Header 236 “Read Opto States”**

Sent byte: none

Received byte: 1 status Byte of optical sensors

Where 1 bit=0 indicates free way

Bit 0: M1

Bit 1: M2

Bit 2: M3 return into hopper

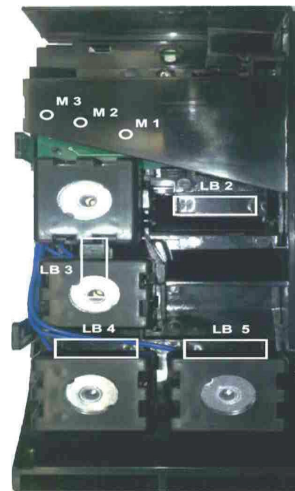
Bit 3: LB5 right storage position

Bit 4: LB4 left storage position

Bit 5: LB2 supply exit coin

Bit 6: LB3 coin recover

Bit 7: NU


**Header 231 “Modify inhibit mask”**

Sent byte: 2 byte. Each bit of 2 byte qualifies or disables the correspondent channel. Bit 0 of the first byte controls the channel 1, bit7 of the second byte controls the channel 16

Received byte: None

The hopper replies with ACK

At the turn on and at the reset moment all the channels are active.

Pay very attention: if some supplied coins get disabled some amount could result impossible (it will happen a timeout in the payment)

**Header 230 “Request inhibit mask”**

Sent byte: None

Received byte: 2. The meaning is the same for the 2 2 byte sent with command 231.

**Header 197 “Calculate ROM checksum”**

Sent byte: none

Received byte: 3 byte [byte1],[byte2],[byte3] that indicate the checksum of the peripheral device. The first byte is always LSB so the checksum is  $[\text{byte1}] + 256 \times [\text{byte2}] + 65536 \times [\text{byte3}]$

#### Header 169 "Request address mode"

Sent byte: none

Received byte: 1 with value =32

Value 32 indicates that the address of the peripheral device is set through deep-sw.

#### Header 164 "Enable Hopper"

Sent byte: 1 byte. If byte=165 the hopper is qualified for payments.

Received byte: None (the hopper replies with ACK)

#### Header 163 "Test Hopper"

Sent byte: none

Received byte: 2 byte that indicate the hopper status

##### Byte 1 (WARNING: the hopper is anyway on duty)

Bit0=1 Stopped hopper for current and restarted after release.

Bit1=1 Timeout during supply (lack of coins or blocked engine)

Bit2=1 Engine has made a release operation during the last supply

Bit3=1 Set bit=1 at PowerOn and reset at first valid command

Bit4=1 Hopper disabled (disable with command 165)

Bit5=1 Hopper stopped from command 132 ("Emergency stop value")

Bit6= 1 Hopper in duty / bit=0 hopper in IDLE (waiting for commands)

Bit7= 1 Problem during coin recovery. Coin not correctly recovered. Bits reset (except bit 3 and 6) at each new qualification (command 164) or at reset (command 1)

##### Byte 2 (ERROR: if only one bit=1 hopper is blocked in "OUT OF ORDER")

Bit0=1 Inductive sensors faulty coins

Bit1=1 Data Checksum wrong coins

Bit2=1 Faulty Hall sensor

Bit3=1 Blocked Hopper for blocked photocells LB2, LB3, M1, M2, M3.

Bit4=1 Blocked Hopper for current limit overtaken after three attempts.

Bit5=1 Blocked Hopper for polling-timeout during "purge hopper"

Bit6,7= NU

**The error bit get reset (and the hopper become active again) if the condition of the error do not exist anymore. Only the bit 5,6 get reset at "power on" (serious error that require an operator).**

#### Header 134 "Dispense hopper value"

There are 2 dispensing modes: the first requires the entry of the amount (in cents), while the second requires the entry of the 8 values corresponding to the number of coins to be dispensed for channels 1 to 8 (channels 9 to 16 are not accessible with this second mode)

Mode 1: to dispense an amount the hopper needs a parameter of 10 bytes:

The first 8 bytes [byte1],...[byte8] must all be = 0

The last 2 bytes sent (the first is LSB) indicate the value in cents to be dispensed, which is therefore [byte4]+256\*[byte5].

**SENDING THE VALUE ZERO REFILLS THE POCKETS. THE HOPPER BEHAVES AS FOR A NORMAL DISPENSING, SO THE STATUS MUST BE FOLLOWED BY POLLING VIA COMMAND 133 BYTE 6 OF WHICH INDICATES WHEN IT RETURNS TO 0 BECAUSE THE POCKETS ARE FULL OR THE SEARCH HAS TIMED OUT DUE TO SUITABLE COINS MISSING**

Mode 2: to dispense an amount the hopper needs a parameter of 16 bytes:

The first 8 bytes [byte1],...[byte8] must all be = 0

Bytes 9 to 16, respectively, indicate the number of coins to be dispensed for channel 1 up to channel 8 (byte9=channel1... byte16=channel8). The pockets are used at start-up only if necessary and are not used any more during that payment

In both modes the hopper always responds with ACK.

The hopper is disabled at each payment. At each payment request without an enabling, the hopper responds with **NAK**. **DURING THE PAYMENT ONLY COMMANDS 133 132 and 1 ARE ADMITTED (the others return NAK). If we send reset (header=1) instead of "Emergency stop value" (header 132), the operation stops but the counters of coins dispensed are reset**

#### Header 133 "Request hopper polling value"

Sent byte: none

Received byte: 6 byte.

The first byte [data1] is an "Event Counter" that is a counter increased for each paid coin (any coin but it has to be paid not rejected).

When 0 is set at reset it starts again from 1 at overflow (0 at reset then cyclic from 1 and 255)

Byte 2 e 3 indicate the value in cent of the value amount that has to be supplied = [data2]+256\*[data3] **that is always=0 given that it does not have meaning on CCS**

Byte 4 e 5 indicate the value in cent of the supplied amount value=[data4]+256\*[data5]

Byte6 indicate the status of the hopper about the payment.

If=7-9 payment in progress (7-> stable engine 8-> forward engine 9->behind engine)

If=0 hopper in idle and last start of the engine has been for command 134

**All the byte except "event counter" get reset at each new command 134.**

**At power off the counters get saved, the event counter instead start again from 0. At a command of reset instead they get all reset.**

#### Header 132 “Emergency stop value”

Sent byte: none

Received byte: 2 byte (the first is LSB) that indicate the value in cent of the paid amount until the stop, so it counts  $[data1] + 256 * [data2]$

The hopper after this command get disabled.

Repeating the command, the value get always repeated and saved until reset, at turn of or at a new command of payment (header 134).

**ATTENTION: the reply depending on the status of the route of the coins can be very delayed and consider a timeout of at least 500ms**

#### Header 131 “Request hopper coin value”

Sent byte: 1 byte that indicate the channel (1-16) corresponding to the coin whose code and value you would like to find out.

Received byte: 8 byte. The first 6 indicate the standard coin code ccTalk for Example “EU200” for 2€. The last 2 byte (the first is LSB) indicate the value in cent and counts  $[data7] + 256 * [data8]$

#### Header 130 “Request indexed hopper dispense count”

Sent byte: 1 byte that indicates the channel (1-16) corresponding to the coin whose quantity of supplied pieces you would like to find out.

Received byte: 2 byte (the first is LSB) that indicate the number of the coin supplied from that channel from the reset. The value is  $[data1] + 256 * [data2]$

**The value of the first 8 counters (channels 1-8) is saved at the turn off while at the command reset they get reset.**

#### Header 121 “Purge hopper”

The hopper get totally empty. Also non-recognize coins get supplied.

This is a service command that must be execute together with an operator because it can be potentially dangerous. For reason of security a precise method has to be used otherwise the command do not execute. Moreover, during the emptying the command 133 must be executed in a polling time not larger than 1 second. If this command is not properly sent the hopper get blocked at first exit coin after the timeout occurred. It has to be turned off and turned on again (reset hardware) in order to restart it.

Method:

- 1- Send a reset (command 1)
- 2- Request of serial number (command 242)
- 3- Qualify the hopper (command 164)
- 4- **NOW WITHIN A SECOND THE FOLLOWING COMMAND HAS TO BE EXECUTED OTHERWISE THE COMMAND WILL NOT BE EXECUTED**
- 5- Send command 121 with the correct serial numbers as for command 134 (first method). The hopper replies with ACK.
- 6- SEND IN POLLING UNTIL COINS TIMEOUT AND FINISH COMMAND 133.

From the value of byte 6 you can figure out if the emptying is finish. If all the coins get (byte2\_3 del command 133=0) the value can be counted from byte 3\_4 of command 133

**DURING THE EMPTYING ONLY THE COMMANDS 133 AND 132 AND 1 ARE ALLOWED (the others will be back to a NAK). If we send a reset (header=1) instead of an “Emergency stop value”(header 132) the operation stops but clearly the counters of the supplied coins will be reset.**

#### Header 31 “Reprogram pocket”

For reprogramming the channel to be used for the pockets.

The reprogramming is valid solely while the pocket is empty and the hopper is disabled. Take great care because if the values are not chosen correctly based on the coins present and the value to be dispensed it may be difficult (or even impossible) to conclude the payment. The simplest way to reprogramme the pockets if they are full is to pay an amount equal to the value of the pockets themselves (which can be found in command 30)

The command is mainly designed to avoid programming values through the PC, allowing them to be run directly on site. This is useful when having to replace the peripheral device with one that has differently programmed pockets.

bytes sent: 2

byte1=number of channel to be programmed in left pocket (value1-16)

byte2=number of channel to be programmed in right pocket (value1-16)

bytes received: 2

byte1=new value programmed in left pocket. If =255 it means it has not been changed

byte2= new value programmed in right pocket. If =255 it means it has not been changed

#### Header 30 “Request pocket status”

For finding out pocket status.

bytes sent: none

bytes received: 7

byte1=number of channel programmed in left pocket

byte2=number of channel programmed in right pocket

byte3\_4=coin value programmed in left pocket (byte3=LSB)

byte5\_6=coin value programmed in right pocket (byte5=LSB)

byte7: bit0=1 indicates that the left pocket is full, bit1 does the same for the right pocket

N.B.: the small coins (programmed from chan6 onwards) store the presence in the pockets in eeprom given that they may not be detected correctly by the photocells. Command 30 is therefore preferable to command 236 for reading the actual presence of the coins in the pockets. For the larger coins (10 cents and up), either command is fine.

#### Header 4 “Request Comms Revision”

Sent byte: none

Received byte: 3 byte [1],[4],[3]

#### Header 1 “Reset device”

Sent byte: none

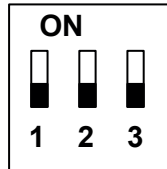
Received byte: none

The hopper replies with ACK after the execution of a soft reset.

## CCS START LIST CONTROLS CSS

The serial address of the hopper can be set using the Dip-Switch group.

The address changes depending on the combination of Dip-Switch, following the table:



Dip-Switch 1	Dip-Switch 2	Dip-Switch 3	N° hopper	serial address
			1	3
		X	2	4
	X		3	5
	X	X	4	6
X			5	7
X		X	6	8
X	X		7	9
X	X	X	8	10

- Command 134 "Dispense hopper value" is used for counting coins. Send any value of payment and the count does not stop until you send a "132-Emergency stop value" or until a "timeout lack of coins" intervenes.
- In order to count the collected coins use a polling command "133-Request hopper polling value" during the count or after that it is finished.
- At the end of the count in order to know the denomination of the collected coins use the "130-Request indexed hopper dispense count". (Attention: the count of the denominations is progressive so it is resettable only at the moment of the starting or with a reset operation).
- CS does not have a "purge hopper" command. Command "134-Dispense hopper value" after a reset (when all the coins are addressed on the sorter variance 0) is able to "empty" the entire hopper.
- It can be programmed until 16 coins in the 16 tubes but only 2 (mod ccs2) or 3 (mod ccs3) plus a variance can be driven where the coins can be send. Programmed coins on the same separator channel will mix.

### Recommended series for coins count:

1- hopper reset (command 1)

2- hopper qualification (command 164)

3-to set (command 210) sorter 1 for the channels that contains coins that have to go left 2 for channels with coins that have to go rights and 3 (for ccs3) for coins on the left.

4-send command 134 with value 0 and if you would like to reset the counters of the coins or 1 if you would like to have a develop of the counters.

5-supervise in polling with command 133 the supplied amount until the timeout of lack-of-coins intervenes (data byte 6=0) or if you would like to stop send the command 132.

6-use the command 130 on the programmed channels in order to find out the denomination of the collected coins.

**ATTENTION:** in case of interruption of power source the counters of the first 8 coins will be saved together with the supplied total.

### Header 254 "Simple poll"

Sent byte: none

Received byte: none

The hopper replies ACK if the command is identified or the address is correct

### Header 246 "Request manufacturer ID"

Sent byte: none

Received byte: String that indicates the builder. In this case "MicroHard.Srl."

### Header 245 "Request equipment category ID"

Sent byte: none

Received byte: String that indicates the kind of peripheral device. In this case "Coin Entry System"

### Header 244 "Request product code"

Sent byte: none

Received byte: string that indicates the kind of product code. In this case "X5 CCS2"

### Header 242 "Request serial number"

Sent byte: none

Received byte: 3 byte [byte1],[byte2],[byte3] that indicates the serial number of the peripheral device. The first byte is always LSB so the serial number is [byte1]+256\*[byte2]+65536\*[byte3]

#### Header 241 “Request software revision”

Sent byte: none

Received byte: string with soft and hardware versions. Ex. “1.0 1.0” indicates software rev 1.0 and hardware rev 1.0

#### Header 236 “Read Opto States”

Sent byte: none

Received byte: 1 status Byte of optical sensors

Where 1 bit=0 indicates free way

Bit 0: M1

Bit 1: M2

Bit 2: M3 return into hopper

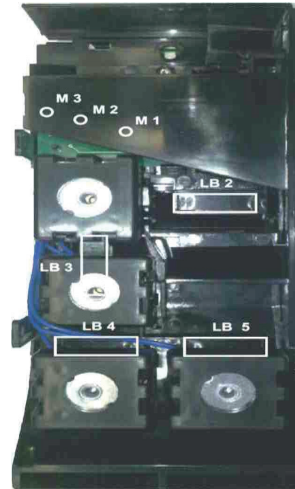
Bit 3: LB5 right storage position

Bit 4: LB4 left storage position

Bit 5: LB2 supply exit coin

Bit 6: LB3 coin recover

Bit 7: NU



#### Header 217 “Request Payout Hi-Lo status”

Sent byte: none

Received byte: 1 byte that indicate the status and the presence of sensors of minimum and maximum

Bit0=1 Coins do NOT overtake the minimum sensor (if=0 they overtake)

Bit1=1 Coins do NOT overtake the maximum sensor (if=0 they overtake)

Bit2=1 Minimum sensor present (if=0 absent)

Bit3=1 Maximum sensor present (if= absent)

#### Header 210 “Modify sorter path”

Sent byte: 2

Received byte: None (hopper replies with ACK)

Byte 1 indicates the channel 1-16 and byte 2 indicates the separation channel where the coin is sent. Channel 0 is the reject channel, channel 1 is the left one and channel 2 is the right one.

If for example the channel 1 is sent to left channel byte 1,1 must be sent.

At reset and at power on all the channels will be driven to the reject. It must be set as the coin changers.

If a channel is driven on the reject during the count (command 134) its value is not damaged. Therefore to programme a coin on the reject channel is like inhibit it.

#### Header 209 “Request sorter path”

Sent byte 1, that indicates the channel where you would like to find out the separation channel.

Received byte: 1 that indicates the requested value.

#### Header 197 “Calculate ROM checksum”

Sent byte: none

Received byte: 3 byte [byte1],[byte2],[byte3] that indicate the checksum of the peripheral device. The first byte is always LSB so the checksum is  $[byte1]+256*[byte2]+65536*[byte3]$

#### Header 169 “Request address mode”

Sent byte: none

Received byte: 1 with value =32

Value 32 indicates that the address of the peripheral device is set through deep-sw.

#### Header 164 “Enable Hopper”

Sent byte: 1 byte. If byte=165 the hopper is qualified for payments.

Received byte: None (the hopper replies with ACK)

#### Header 163 “Test Hopper”

Sent byte: none

Received byte: 2 byte that indicate the hopper status

##### Byte 1 (WARNING: the hopper is anyway on duty)

Bit0=1 Stopped hopper for current and restarted after release.

Bit1=1 Timeout during supply (lack of coins or blocked engine)

Bit2=1 Engine has made a release operation during the last supply

Bit3=1 Set bit=1 at PowerOn and reset at first valid command

Bit4=1 Hopper disabled (disable with command 165)

Bit5=1 Hopper stopped from command 132 (“Emergency stop value”)

Bit6= 1 Hopper in duty / bit=0 hopper in IDLE (waiting for commands)

Bit7= 1 Problem during coin recovery. Coin not correctly recovered. Bits reset (except bit 3 and 6) at each new qualification (command 164) or at reset (command 1)

## **Byte 2 (ERROR: if only one bit=1 hopper is blocked in "OUT OF ORDER")**

Bit0=1 Inductive sensors faulty coins  
 Bit1=1 Data Checksum wrong coins  
 Bit2=1 Faulty Hall sensor  
 Bit3=1 Blocked Hopper for blocked photocells LB2, LB3, M1, M2, M3.  
 Bit4=1 Blocked Hopper for current limit overtaken after three attempts.  
 Bit5=1 Blocked Hopper for polling-timeout during "purge hopper"  
 Bit6,7= NU

**The error bit get reset (and the hopper become active again) if the condition of the error do not exist anymore. Only the bit 5,6 get reset at "power on" (serious error that require an operator).**

## **Header 134 "Dispense hopper value"**

After the qualification of the hopper send only the parameter for the count.

If we send as parameter 0, the counters will be reset and they restart from 0, if we send 1 the counters will be progressive.

Received byte: none. The hopper replies with ACK if correct command and previously qualified with command 164.

At each payment the hopper get qualified. To a request of payment without qualification the hopper replies with NAK.

**During payment only the commands 132 and 132 and 1 are allowed (the others get back to a NAK) If we send the reset (header=1) instead of an "Emergency stop value" (header 132) the operation stops anyway but clearly the count of the supplied coins will be reset.**

## **Header 133 "Request hopper polling value"**

Sent byte: none

Received byte: 6 byte.

The first byte [data1] is an "Event Counter" that is a counter increased for each paid coin (any coin but it has to be paid not rejected).

When 0 is set at reset it starts again from 1 at overflow (0 at reset than cyclic from 1 and 255)

Byte 2 e 3 indicate the value in cent of the value amount that has to be supplied =  $=[data2]+256*[data3]$  **that is always=0 given that it does not have meaning on CCS**

Byte 4 e 5 indicate the value in cent of the supplied amount  $value=[data4]+256*[data5]$

Byte6 indicate the status of the hopper about the payment.

If=7-9 payment in progress (7-> stable engine 8-> forward engine 9->behind engine)

If=0 hopper in idle and last start of the engine has been for command 134

**All the byte except "event counter" get reset at each new command 134.**

**At power off the counters get saved, the event counter instead start again from 0. At a command of reset instead they get all reset.**

## **Header 132 "Emergency stop value"**

Sent byte: none

Received byte: 2 byte (the first is LSB) that indicate the value in cent of the paid amount until the stop, so it counts  $[data1]+256*[data2]$

The hopper after this command get disabled.

Repeating the command, the value get always repeated and saved until reset, at turn of or at a new command of payment (header 134).

**ATTENTION: the reply depending on the status of the route of the coins can be very delayed and consider a timeout of at least 500ms**

## **Header 131 "Request hopper coin value"**

Sent byte: 1 byte that indicate the channel (1-16) corresponding to the coin whose code and value you would like to find out.

Received byte: 8 byte. The first 6 indicate the standard coin code ccTalk for Example "EU200" for 2€. The last 2 byte (the first is LSB) indicate the value in cent and counts  $[data7]+256*[data8]$

## **Header 130 "Request indexed hopper dispense count"**

Sent byte: 1 byte that indicates the channel (1-16) corresponding to the coin whose quantity of supplied pieces you would like to find out.

Received byte: 2 byte (the first is LSB) that indicate the number of the coin supplied from that channel from the reset. The value is  $[data1]+256*[data2]$

**The value of the first 8 counters (channels 1-8) is saved at the turn off while at the command reset they get reset.**

## **Header 4 "Request Comms Revision"**

Sent byte: none

Received byte: 3 byte [1],[4],[3]

## **Header 1 "Reset device"**

Sent byte: none

Received byte: none

The hopper replies with ACK after the execution of a soft reset.


**CONTROLS cctalk SUPPORTED by DPO and CCS**

Header 254	Simple poll	
Header 253	Address poll	
Header 246	Request manufacturer id	
Header 245	Request equipment category id	
Header 244	Request product code	
Header 242	Request serial number	
Header 241	Request software revision	
Header 236	Read opto states	
Header 231	Modify inhibit mask	* only for DPO
Header 230	Request inhibit mask	* only for DPO
Header 217	Request payout high / low status	
Header 210	Modify sorter path	** only for CCS
Header 209	Request sorter path	** only for CCS
Header 197	Calculate ROM checksum	
Header 169	Request address mode	
Header 164	Enable hopper	
Header 163	Test hopper	
Header 134	Dispense hopper value	
Header 133	Request hopper polling value	
Header 132	Emergency stop value	
Header 131	Request hopper coin value	
Header 130	Request indexed hopper dispense count	
Header 121	Purge hopper	* only for DPO
Header 031	Reprogram pocket	*only for DPO
Header 030	Request pocket status	*only for DPO
Header 004	Request comms revision	
Header 001	Reset device	



## 5 SPARE PARTS


 Use exclusively original spare parts to replace any components.

 Use of non-original and/or non conform parts (if not authorized exclusively by the assistance center in writing) release the manufacturer from all liability.

To request spare parts, photocopy the page of the pertinent spare parts table and fill out the table completely, indicating the table containing the part, its reference number on the drawing and the quantity of parts requested, and your details.

Requests lacking the above data will not be taken into consideration.

Send the copy/ies by fax to the number +39 0547 81247

  
**SPARE PARTS REQUEST FORM**  
 Send copy by fax to the number +39 0547 81247

**CUSTOMER DATA**

Company name \_\_\_\_\_

Address \_\_\_\_\_ Date of request \_\_\_\_\_

City/Town \_\_\_\_\_

Tel. \_\_\_\_\_ Stamp/Signature \_\_\_\_\_

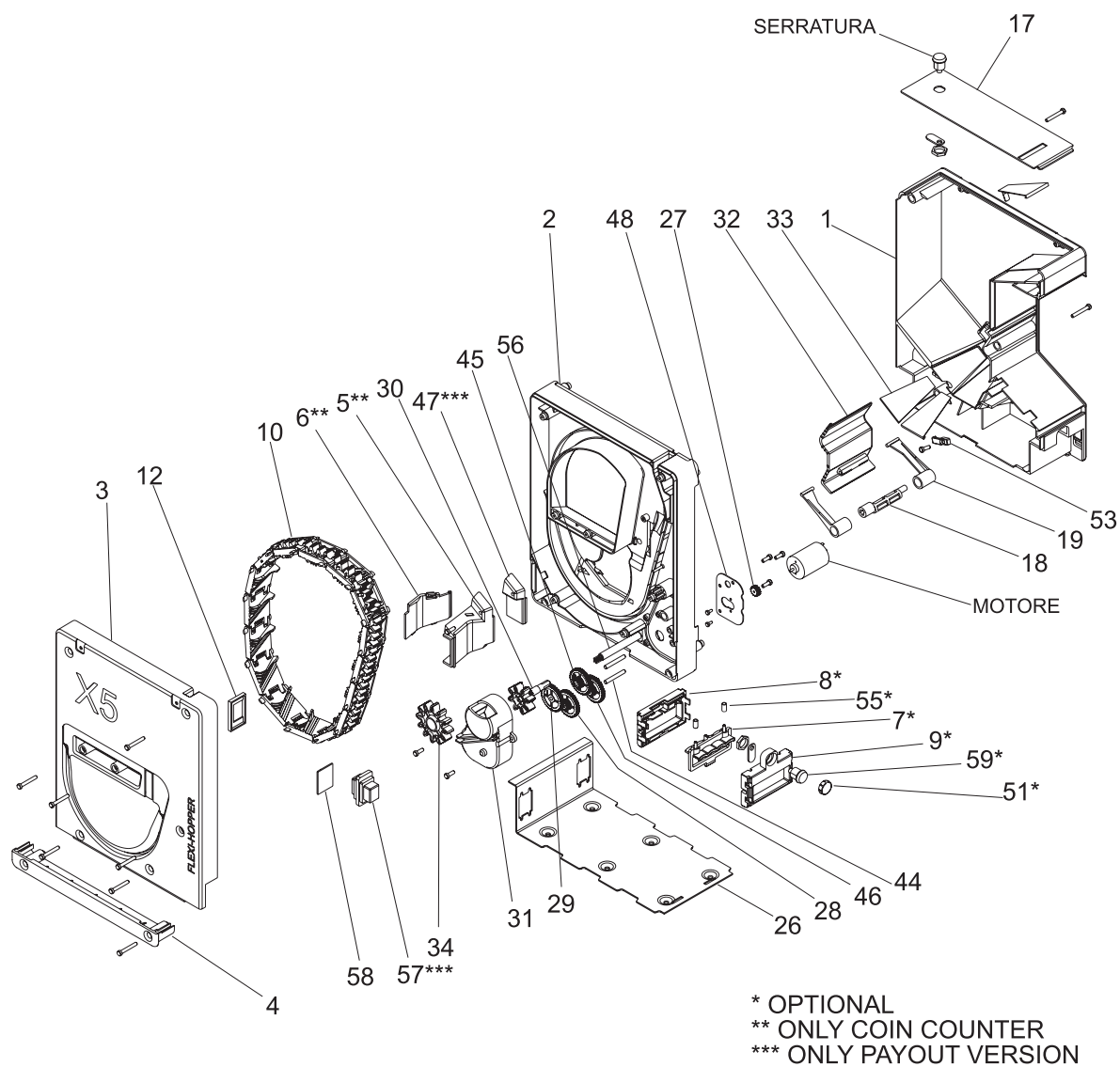
Fax \_\_\_\_\_

TABLE	NUMBER	QUANTITY

TABLE	NUMBER	QUANTITY

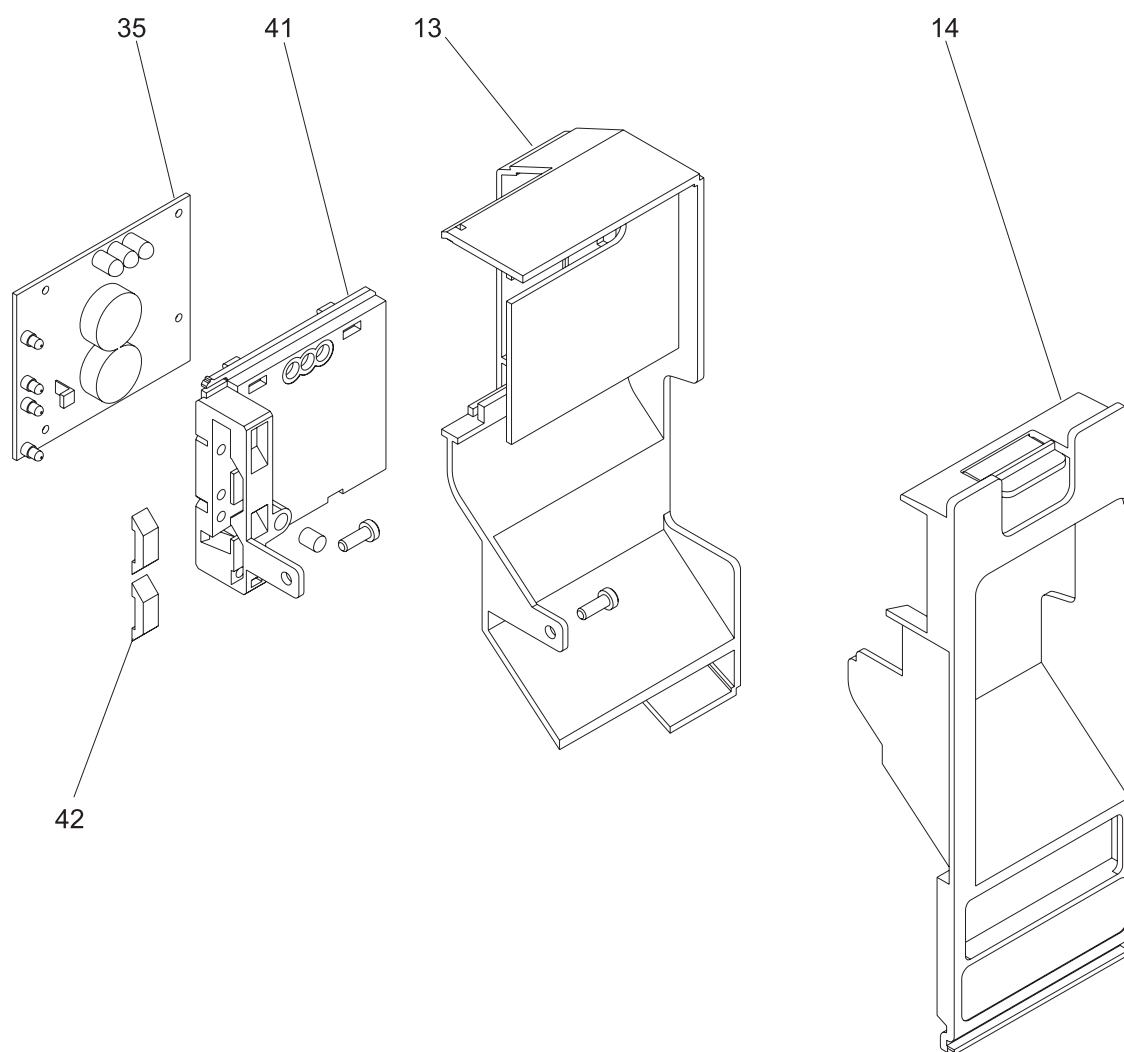
TABLE	NUMBER	QUANTITY



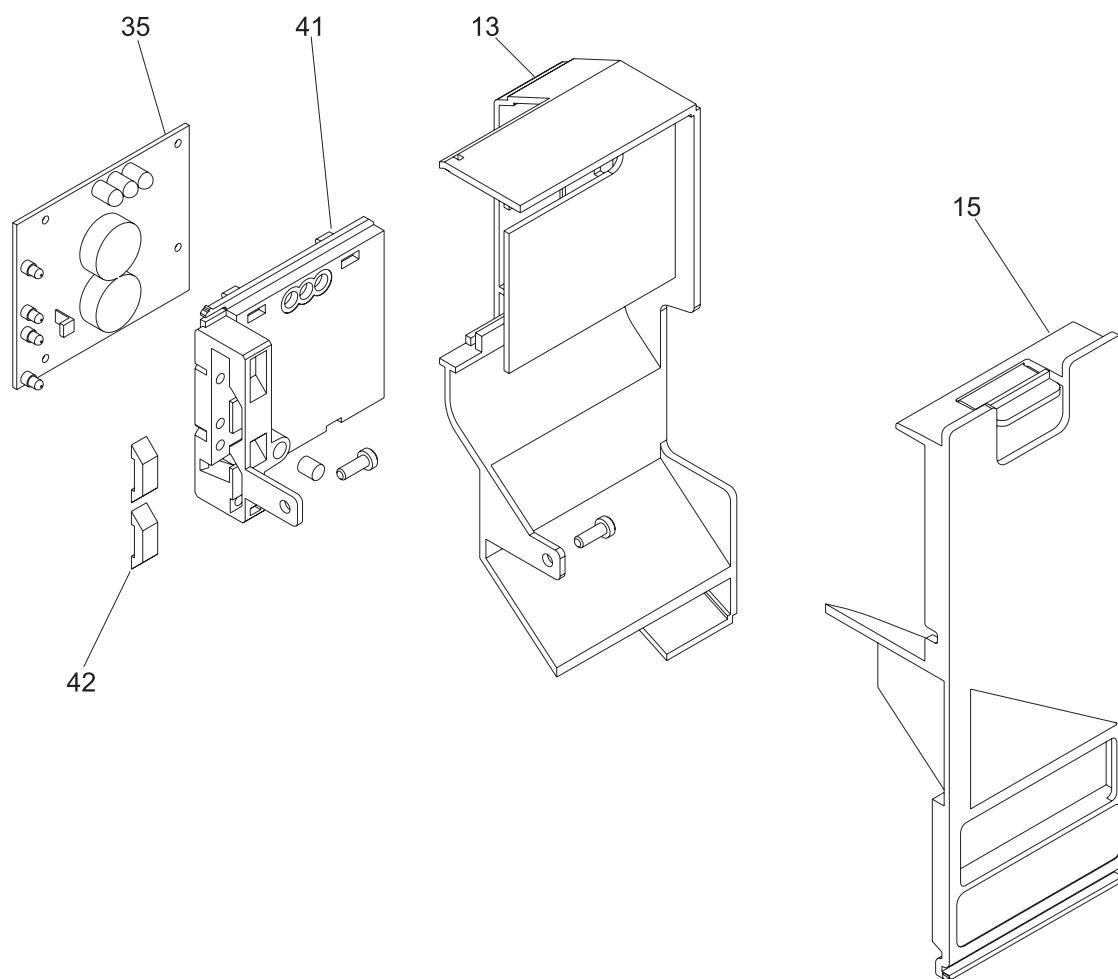


**TAV. 1 - PAYOUT PART CINCH CONNECTOR**





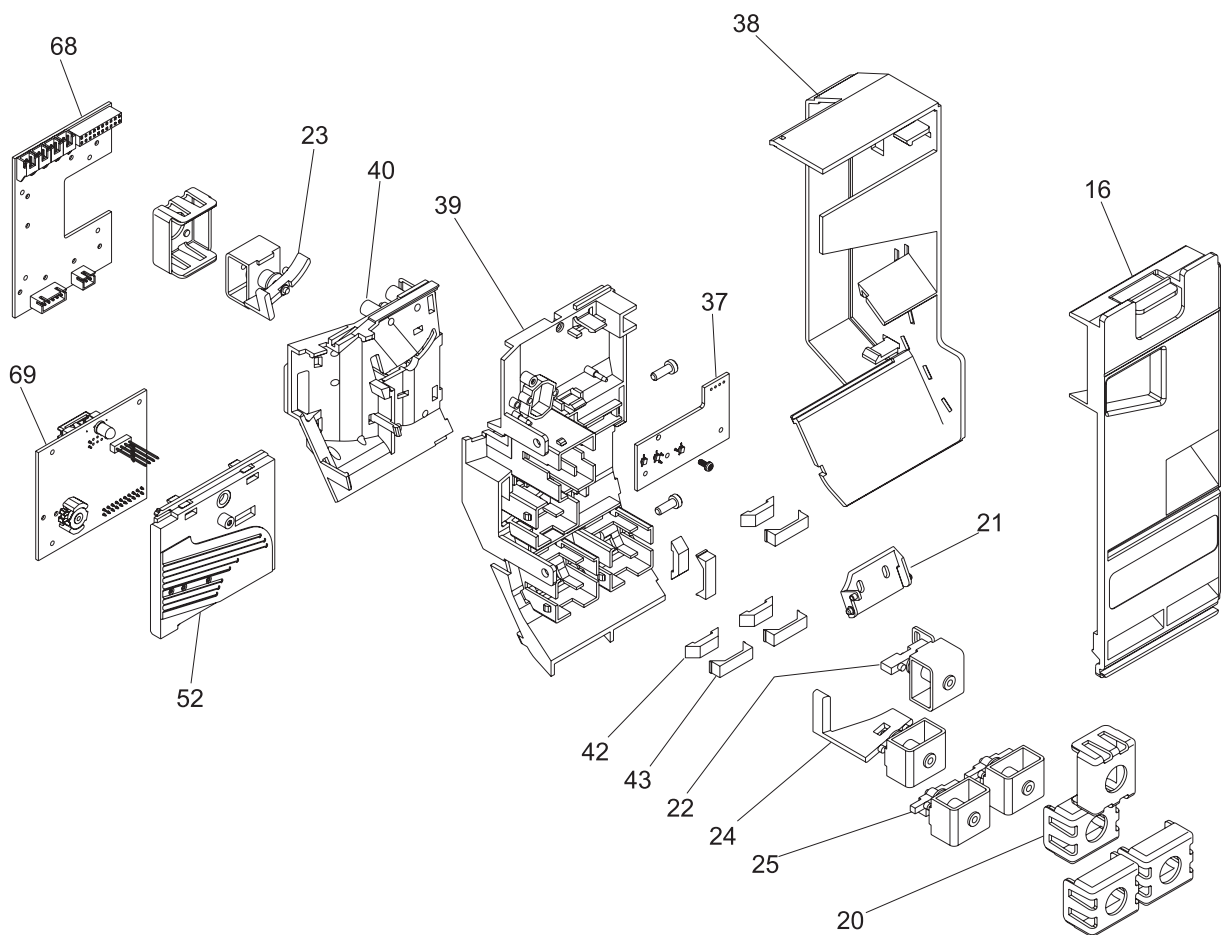
**TAV. 3 - STANDART VERSION**



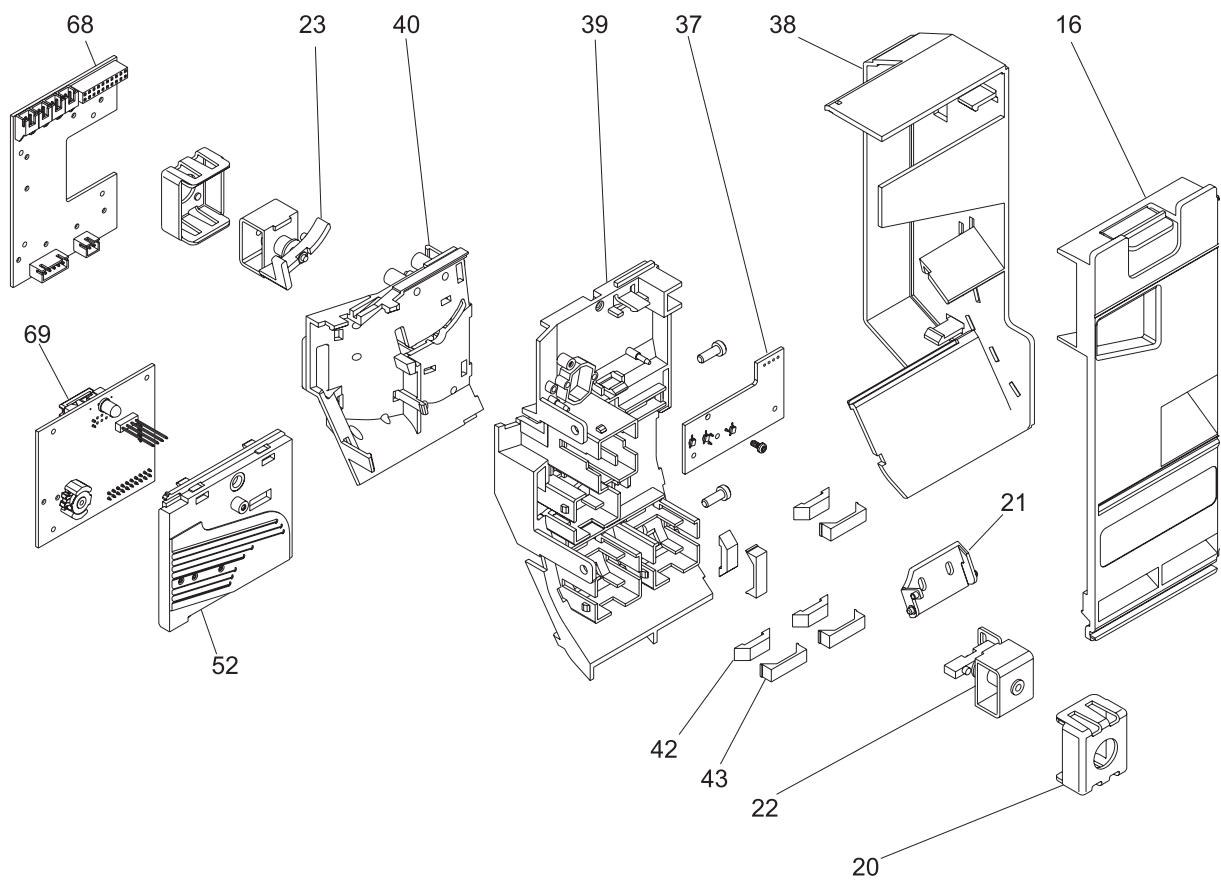
**TAV. 4 - SIDE VERSION 1**



**TAV. 4a - SIDE VERSION 2**



**TAV. 5 - DYNAMIC PAY OUT**



**TAV.6 - COIN COUNTER AND SORTER**



**Microhard s.r.l.**  
**Via dei Platani. 7 - 47042 Cesenatico (FC)**  
**TEL.: 0039-0547 75450 FAX: 0039-0547 81247**  
**[info@microhard.it](mailto:info@microhard.it)**  
**[www.microhard.it](http://www.microhard.it)**