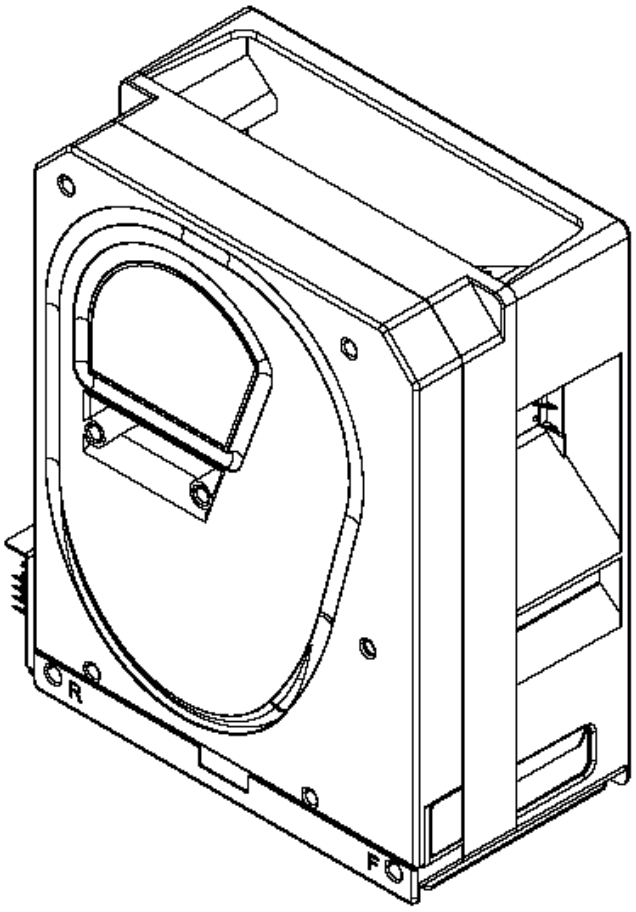
Evolution

Evolution Hopper EV2000, EV3000 and EV4000

Product Manual

Version 1.2 / Oct 2009



SUZO HAPP®

GAMING, AMUSEMENT AND INDUSTRIAL COMPONENTS

Revision History

Revision	Date	Comment	By
0.1	5 sept 2005	Initial Release	R.T.
0.2	12 sept 2005	Added hopper application information (chapter 7) Renumbered chapters	R.T.
0.3	18 oct 2005	EV1001 -> EV2000 and EV1002 -> EV3000 SUH2 -> SUH1 for compatibility reasons	R.T.
0.4	22 may 2006	EV4000 added	R.T.
0.5	25 oct 2006	Added hopper Empty command	R.T.
1.0	27 mrt 2006	Drawing	E.S.
1.1	22 oct 2007	Belt partnr, exploded view	E.S.
1.2	8 oct 2009	Text corrections item 3.9 and 6.4	E.S

This manual is intended only to assist the reader in the use of this product and therefore Suzo International shall not be held liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any omission from this manual or any incorrect use of the product.

Design and specifications are subject to change without notice.
Wijzigingen in ontwerp en technische gegevens voorbehouden, zonder kennisgeving.
La conception et les spécifications sont modifiables sans préavis.
El diseño y especificaciones están sujetos a cambios sin previo aviso.

WARNING!

Failure to observe the interface requirements specified in this technical manual may result in miscounts, damage to the electronics and the motor of the hopper or create unacceptable voltage drops, affecting other units depending on the same power supply.

Contents

Tables

Figures

1. Introduction

The Suzo-Happ group has now introduced it's own version of a belt driven hopper.
After the successful Cube hopper, the Gold series, the Excel Casino Hoppers and the Escendo escalator hopper has Suzo-Happ made it's own improved version of this unique hopper concept.
Easier service-ability and higher speed are the two key elements for developing this product.
This product is compatible with most other belt driven hoppers in the market.

2. Safety Note

To meet the requirements for EN 60950 the equipment must be installed according to the following requirements: The equipment must be protected by a 3A fuse.
The equipment must be supplied from a SELV limited power source.
The equipment must be installed in an enclosure but positioned so that it is external to any fire enclosure area within the main enclosure.

3. General Description

3.1 Description

The Evolution Hopper is an universal intelligent large capacity coin and token dispenser ideal for a wide range of applications including Gaming, Vending and Transportation systems.
The Evolution hopper will handle most coins in the range 16.25mm to 30mm diameter and 1.25mm-3.5mm thick, giving the following approximate capacities:

$$\text{Capacity} = \text{Hopper volume} / \text{Coin volume} = \frac{1{,}200{,}000}{\frac{\pi \times D^2}{4} \times T}$$

D = Coin diameter (mm)
T = Coin thickness (mm)

Diameter (mm)	Thickness (mm)	Coin type	Approx. capacity
25.75	2.20	2 Euro	1000
23.25	2.35	1 Euro	1200
24.25	2.40	0.50 Euro	1100
24.25	1.75	US quarter	1500

Table 1: Hopper capacity for some popular coins

The Evolution hopper standard can handle coins between 21.01 and 30.00 mm.
A Euro coin track is available for all euro coins (between 19.00 – 26.40 mm).
A small coin track is available for smaller coins between 16.25 and 20.90 mm (optional).
A large coin track is in preparation for coins of 31 mm (optional).
The payout speed depends on the coin size and the amount of coins in the hopper but the average speed is approximately 4 coins per second.
Precise payout is ensured through optical sensing and verifying of coin dispensing with an electronic security signal which alerts against coin jams, failed sensors and a bad power supply.
LED indicators are provided for easy visual checking of power supply, security status and coin sensors.

The ccTalk interface can be specified with 3 different protocols:
- EV2000 (SUH1-USE_SERNR): Uses the Italian version of ccTalk.
- EV3000 (SUH1) : Uses the original MC Encrypted version of ccTalk.
- EV4000 (SUH1-NOENCRYPT): Uses the original MC Non-Encrypted version of ccTalk.

See section Enable Hopper (header 164) on page 25 for a detailed explanation between the different models.

3.2 Optical Sensors

Optical sensors are fitted on the optic board in the exit window to detect coin payout.
A debounced coin output is available on pin 3 and pin 11. When no coins are present at the exit window, the optical sensors are clear, the output transistors are open circuit, and the LED indicator is off. Coins passing the optical sensors obstruct the light path causing the output transistors to pull down to OV and the GREEN LED SENSOR indicator switches on.

3.3 Optical Security Feature

The output of the optical sensor is monitored by the microprocessor and if the sensor remains obstructed for more than one second, the motor will be braked and will remain off until either the sensor is cleared or power down takes place.

3.4 Motor Current Limit

The motor current is monitored by the processor. When the motor initially starts, the current is build up gradually using PWM-current control. This reduces the high initial surge currents that occur in non-current controlled motor driver circuits.
If the current rises above a preset value, then a jam is deemed to have occurred. The motor is braked for 50ms then reversed for 150ms. After a further 50ms braking, the motor is started in the forward direction again.
The current is tested after 100ms and if the jam has not been cleared the reversing cycle will be repeated. This action will continue until the jam has cleared. This reversing action is effective in clearing soft jams.
One further action is to test the current in the reverse direction during the final 50ms of the reversing cycle. If during that time period an over current is detected, then the motor will be braked for 50ms and then disabled for 1 second. This action limits the duty cycle sufficiently in the case where a jam is solid in order to prevent motor damage.

3.5 Coins With Holes

The Evolution hopper will work with most coins/tokens with holes depending on the size of the hole in relation to the diameter. To make sure whether your desired coin is qualified within the specifications of the Evolution Hopper please contact the Suzo-Happ technical department
The exit window has been designed so that more coins with holes will be counted correctly.
No adjustments are necessary to cope with standard and small coins.

3.6 High Security Exit Window

3.6.1 Description

The payout window uses optics consisting of an IR-Led transmitting a beam that is reflected by a prism in an U-shaped form and received back on an IR-receiver.
The intensity of the IR-pulse is minimized, so that the (somewhat transparent) plastic coins as well as highly reflective coins are detected as optimal as possible.
The intensity of the IR-pulse is adjusted dynamically to a higher level if the opto-sensor becomes dirty.

3.6.2 Security

- The IR-led transmits pulses with a random duty-cycle between 20 – 25%.
- When light is received when no IR-pulse is being transmitted, (exit window is 'blinded' by external light), the hopper will stop immediately if it was running, the security led will go off and the security output will go high (error state).
- If the opto-sensor is interrupted for more than 1 second, the hopper will also stop and go into error state. If the coin exit becomes unblocked again and the hopper start conditions are still met, the hopper will resume running.

3.7 Options

3.7.1 Track options

The standard Evolution Hopper handles coins in the diameter range of 21.01 – 30.00 mm,
A Euro coin track is available for all euro coins (between 19.00 – 26.40 mm).
The small coin Evolution Hopper handles coins in the diameter range of 16.25 mm 20.90 mm.

3.7.2 Connector position

The 12pin connector can be in one of two positions, either on the opposite side of the coin exit, known as the Rear (R) position, or on the same side as the coin exit, known as the Front (F) position.
Standard the Evolution hopper is supplied with the connector on the adjacent position.
The user can easily change this on a Evolution hopper by loosen two screws on the bottom section and take out this part than place the cable with the connector at the opposite side.

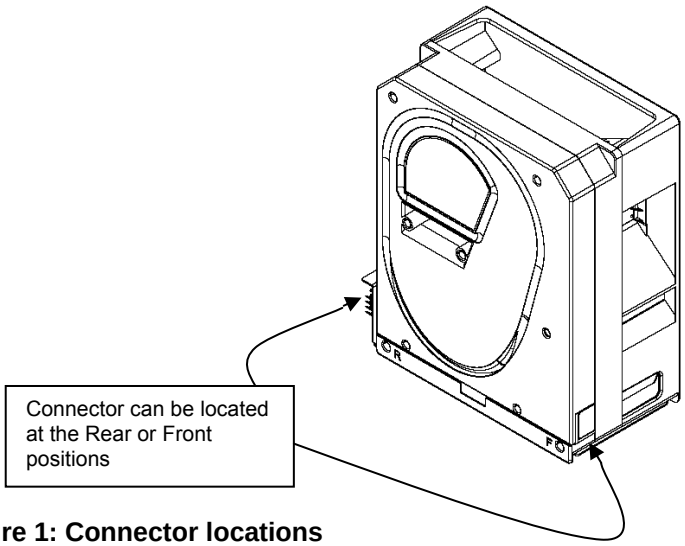


Figure 1: Connector locations

3.8 Level Sensing

All Evolution Hoppers are standard supplied with a low level and high level sensing function.

3.9 Coin Sizes

Track type	Coin sizes	Color	Art. nr.
Standard (€2, €1, €0.50, €0.20)	21.01 – 30.00 mm x 1.25 – 3.30 mm	Red	EV0050
Euro (€2, €1, €0.50, €0.20, €0.10, €0.05)	19.00 – 26.40 mm x 1.50 – 2.50 mm	Yellow	EV0050-3
Euro small (€0.10, €0.05, €0.02, €0.01)	16.25 – 20.90 mm x 1.00 – 3.10 mm	Green	EV0050-4

Table 2: Coin size vs Track type

3.10 Base plate

The base plate offers easy slide in and out function with a pre-fitted connector that can easily be removed for fitting it in a cable-harness.
The base plate is standard supplied with the Evolution hopper.
See Figure 9: Base plate dimensions.

4. Installation

***Important**: Shut-off the power from the host machine until any installation work is completed.*

4.1 Base plate

1. Secure the base plate in position, using the six fixing holes. The hole positions are shown in Figure 9: Base plate dimensions.

2. Wire up the base plate connector to the host machine see Figure 4: Connector for connector details, and chapter 6 for interfacing recommendations.

NOTE: The wire to be used should have a maximum length of 3 meters, and must be capable of handling the maximum currents and voltages specified in Table 13: Electrical Interface.

3. Slide the hopper into the base plate and ensure that the two halves of the connector are securely mated.

4. Turn on the power.

4.2 Safety

1. Do not put a hand into the hopper while the motor is running.
2. Static. It is possible for coins paid out to have a static charge on them.
3. Coins should be discharged to earth before being presented to the user.

5. Mechanical Description

5.1 General

The hopper is mounted in a machine via the base plate.
Electrical connection to the hopper is made via the 12 pin socket on the base plate which mates with the corresponding plug on the hopper body. Coins are stored in the cashbox section of the hopper and fed onto the elevator belt via a passage in the centre plate. The cutout in the centre plate has been designed to regulate the flow of coins onto the belt. The stirrer agitates the coins in the coin box in order to minimize the occurrence of bridging. The elevator belt is driven by a motor, gearbox, and idler gear. Coins are picked up at the bottom of the belt and carried up to the exit window. Optical sensors in the exit window detect the coins as they roll out of the hopper.
A cable connects the main control board to the 12 way socket and carries all power supplies and control signals.

5.2 Removal of the Electronics and Opto Sensor Board.

All the electronics and sensors are placed on one board located behind the exit door at the side of the hopper. Slide the yellow button to the opposite position and remove the exit door were the electronics are mounted to. Take out the board for cleaning the optic sensors is a matter of seconds.
Warning: be care full by re-inserting the board back in the hopper not to damage the cable located at the back of the board!

5.3 Track guard Removal and Refitting

Firstly, locate cut away slots in Centre plate and End plate at the base of the track guard opposite the PCB. Push track guard up to reveal a gap between body moulding and the guard. Insert broad flat bladed screwdriver or equivalent into gap and gently lever out the guard until the leading edge is above the outside edge of the body mouldings. Now slide the guard down towards the cut out and gradually withdraw it. Slide back the track guard to refit.

6. Interface Description

6.1 Serial Communication: Why and how to use it

This section describes some differences between a conventional build machine and a machine using a network like ccTalk.

Serial Communication: One (cheap) network cable does it all
Traditional build machines have a central control board that controls all attached devices.
Each hopper is connected to the board with it's own 4 wire cable and connectors.
Money acceptors are connected to the board using flat cables of 10 wires or more with their own connectors.
See Figure 2: Traditional machine wiring

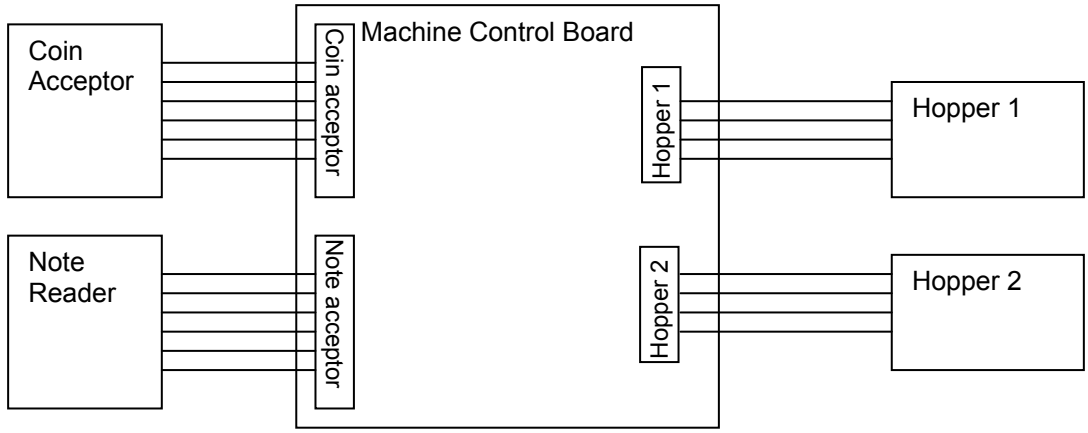


Figure 2: Traditional machine wiring

From Figure 2 can be seen that you need often 3 different cables (1 coin acceptor, 1 note acceptor and 2 hoppers) in order to build a system. This is a quite expensive system.

Figure 3 shows how a system looks if a ccTalk network is used.

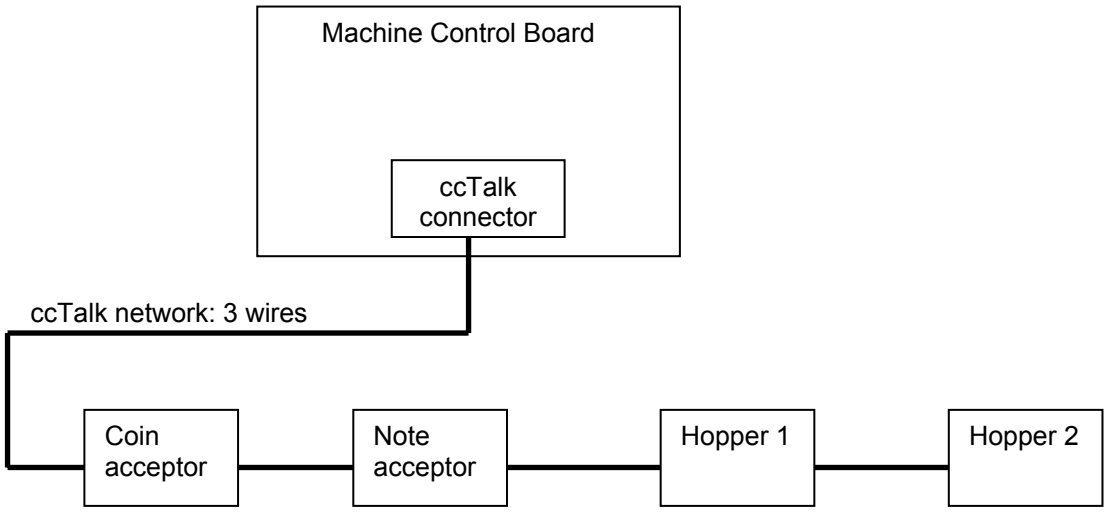


Figure 3: Network wired machine

As can be seen from Figure 3, a networked solution is much simpler, because only 1 long cable consisting of 3 wires (power, data, and ground) connect all devices to the control board. Each device would typically use the same connector.

Serial Communication: More control over the devices

The machine controls the devices by sending data over the network, just like the famous RS232 communication network. The data consists of bytes that are grouped together to form ccTalk messages. Each device has it's own set of command messages and has it's own unique ccTalk address. If the machine sends a message to a hopper, it's will put the hopper's address into the message and send it over the network to the hopper.
The ccTalk protocol defines all messages that are used to communicate with all devices.
There are messages to start the hopper, stop the hopper, return coin level, etc.
So much more control over each device is achieved by using device control messages.

Serial Communication: Fraud elimination?

The traditional way of hopper fraud would be to drill a hole into the machine cabinet and put 24V on the proper wire of the hopper cable.
With a network, each device is controlled using serial data communication. Now you need a PC and knowledge about the ccTalk command messages to start a hopper.
In order to make hopper fraud very difficult, some hopper manufacturers use sophisticated encrypted pin codes in their commands in order to start a hopper. Others use simple codes in their commands to start a hopper.

Serial Communication: Hopper responsibilities

Each ccTalk hopper is equipped with a microcontroller implementing the ccTalk protocol.
You have to define exactly what the hopper should do when it receives a command.
If the hopper receives a payout command, the hopper should start a payout only if the pin code is correct. When it has paid the requested amount of coins it should stop automatically. It is the responsibility of the hopper to stop when the requested amount of coins is paid.

Serial Communication: Bookkeeping

Some hoppers also count the number of coins paid and the number of coins unpaid and even have short term and long term running counters. However is it practical to store all these numbers into the hopper? The machine itself also has it's own copy of all running counters. How do you maintain hopper data and machine data consistent? What happens with the data during a system reset or a power down event? In general only the number of coins paid is the one that is most practical. If the hopper is commanded to pay 10 coins and a power down event occurs, then after the power has returned, the host machine should check how many coins the hopper has paid just before the power failed. The machine should check this number with it's own records and take appropriate action.
During a power down, use the Emergency Stop command to stop the hopper if possible. This command returns the number of unpaid coins for the current payout. Store this result in the memory of the machine.
For hopper maintenance, the total number of coins dispensed during it's life would also be a useful number.

6.2 ccTalk implementation on the Evolution Hopper

The protocol conforms to ccTalk b96.p0.v12.a5.d0.c8.m0.x8.i1.r4 and b96.p0.v24.a5.d0.c8.m0.x8.i1.r4.

9600 baud
open-collector
+12V .. +24V nominal supply
+5V data pull-up
supply sink
connector type 8
slave device
8-bit addition checksum
no encryption
ccTalk minor release 1
ccTalk major release 4

6.3 Connector

Evolution Hopper is only available with the Industry compatible connector. (compatible with the green CINCH connector)

Pin	Description
1	Power supply 0 Vdc
2	Power supply 0 Vdc
3	Coin exit output (not mandatory for ccTalk)
4	Address select 1
5	ccTalk data line
6	High or Top level sense output (not mandatory for ccTalk)
7	Low level sense output (not mandatory for ccTalk)
8	Address select 2
9	Power supply 24Vdc
10	Power supply 24Vdc
11	Coin exit output (not mandatory for ccTalk)
12	Address select 3

Figure 4: Connector pin out

6.4 Device Address

All Evo Hoppers leave the factory with **address 3**. Other addresses can be set by wiring the address select lines. See Table 3: Address select pins.

The address can be changed with serial commands. Unless you have an application requiring more than one Evo hopper on the serial bus, it is strongly recommended you leave the address alone. The default addresses for coin acceptors and bill validators have been made different and will not clash with the Evo hopper. When the hopper is powered up or reset, the hopper will always revert back to it's physical address. Other addresses can be set by commands.

For applications requiring more than one hopper on the serial bus, one or more of the address select lines may be connected to +Vs. A total of 8 unique bus addresses may be generated in this way, in the range 3 to 10 inclusive.

X = Connect to +Vs (Pin 9, 10)			**Serial Address**
Address select 3	**Address select 2**	**Address select 1**	
			3
		X	4
	X		5
	X	X	6
X			7
X		X	8
X	X		9
X	X	X	10

Table 3: Address select pins

All hopper models are also equipped with a small dipswitch located inside the hopper.
Open the PCB cover by sliding the yellow button up to access the dipswitch.
This dip switch can also be used to set the hopper address, instead of the wiring method on the connector. See table below.

Dip Switch 1	**Dip Switch 2**	**Address**
OFF	OFF	3
ON	OFF	4
OFF	ON	5
ON	ON	6

Table 4: Address Dipswitch setting

6.5 Command / Response frame format

Dest. Addr	Nr Data Bytes	Source Addr	Header	Data 1	. . .	Data N	Checksum

The destination address (host address) is usually 1.
The source address (hopper address) starts from 3.

Most responses from the slave have a ACK Header byte followed by zero or more data response bytes. If a command cannot be executed, a NAK Header (Hex 5) will be returned. For example the hopper dispense command which may return a NAK (Hex 5) if the dispense procedure could not be done (for example hopper disabled, busy, pin code not correct, opto-error, etc).

The ACK byte in the responses has value 0.
The Checksum is calculated such that the 8-bit addition (modulus 256) of all bytes in the message from the start to the checksum itself is zero.

6.6 Hopper Setup and Initialization commands

Before the hopper can be used, a number of initialization steps have to be done.

Check communication
Transmit the SIMPLE POLL command to the hopper to check if it is responding with an ACK message:

Simple Poll (header 254)

```
Command : 03 00 01 FE FE
Response: 01 00 03 00 FC
```

Here a normal ACK is received. Device at address 3 is communicating ok.
If no response is received then check the hopper address by sending the ADDRESS POLL command:

Address Poll (header 253)

```
Command : 00 00 01 FD 02
Response: 03
```

From the response can be seen that there is only 1 device on the bus with address 3.
This command returns all device addresses of the devices on the ccTalk bus. If each device has an unique address, then no communication clashes will occur. If a clash occurs, then some devices share the same address. (Address clashes should not occur after power up, since each device will revert to it's default address, which should be unique for every device used. See Table 3: Address select pins). Use the ADDRESS CLASH command (header 252) to check which address clashes. If an address clash occurs, the devices that clashes can be given a random address by using the ADDRESS RANDOM (header 250) command. Send the ADDRESS CLASH again to resolve any other address clashes. Once every device has an unique address, the addresses can be optionally changed to new addresses using the ADDRESS CHANGE (header 251) command.

Once communication is ok, device details can be requested. For your reference all commands are listed below. Most of the commands are not mandatory to operate the hopper.

Request comms revision (header 004)

```
Command : 03 00 01 04 F8
Response: 01 03 03 00 31 33 32 63
```

The 3 bytes(31 33 32) in the response have the following meaning:
[ccTalk level] [major revision] [minor revision]. In our example: 1 3 2.

Request comms status variables (header 002)

```
Command : 03 00 01 02 FA
Response: 01 03 03 00 00 00 00 F9
```

The 3 data bytes (00 00 00) in the response have the following meaning:
[rx timeouts] [rx bytes ignored] [rx bad checksums]
This data can be used to test the quality and load of a ccTalk network.

Clear comms variables (header 003)

```
Command : 03 00 01 03 F9
Response: 01 00 03 00 FC
```

This command is used to reset the comms status variables to 0.

Reset device (header 001)

```
Command : 03 00 01 01 FB
Response: 01 00 03 00 FC
```

This command resets (software reset) the hopper, after the transmission of the response message.
During software resetting (about 50ms) the hopper will not respond to commands.
After the reset, all hopper variables (including status flags) will be reset to their default values.
⇒ A power-up reset (hardware reset) takes about 500 ms, during which the hopper will not respond to any commands.

Request Manufacturer id (header 246)

```
Command : 03 00 01 F6 06
Response: 01 0D 03 00 53 75 7A 6F 20 49 6E 74 20 28 4E 4C 29 E8
```

The data bytes in the response (53 75 7A 6F 20 49 6E 74 20 28 4E 4C 29) give the manufacturer: "Suzo Int (NL)". Remember that the header byte in the response, here 00, means that an ACK is received.

Request Equipment Category id (header 245)

```
Command : 03 00 01 F5 07
Response: 01 06 03 00 50 61 79 6F 75 74 74
```

The data bytes in the response (50 61 79 6F 75 74) stand for "Payout".

Request Product Code (header 244)

```
Command : 03 00 01 F4 08
Response: 01 04 03 00 53 43 48 32 E8
```

The data bytes in the response (53 43 48 32) mean "SUH1", meaning **S**uzo **U**niveral **H**opper MK**1**
The following hopper configurations can be obtained from factory:

Product Code	Description
SUH1	8 encrypted security bytes must be sent with the dispense command
SUH1-NOENCRYPT	8 dummy bytes must be sent with the dispense command
SUH1-USE_SERNR	The hopper serial number must be sent with the dispense command

Table 5: Hopper Product Codes

Always check the Product Code first to determine the hopper operation mode. See also 6.7 Hopper Dispense coins procedure.

Request Serial Nr (header 242)

```
Command : 03 00 01 F2 0A
Response: 01 03 03 00 4E 46 05 60
```

The 3 data bytes are the serial number (hex): 4E 46 05. This is decimal 345678.
The least significant bytes are transmitted first.
If the hopper Product Code is SUH1-USE_SERNR, then be sure to send this serial number along with the dispense command in order to start a payout. See also Dispense Hopper coins (header 167).

Request Software Revision (header 241)

```
Command : 03 00 01 F1
Response: 0B 01 09 03 00 43 75 62 65 20 56 31 2E 36 6E
```

The response data byte (43 75 62 65 20 56 31 2E 38) mean "Evo V1.8".

Request data storage availability (header 216)

```
Command : 03 00 01 D8 24
Response: 01 05 03 00 02 04 08 03 08 DE
```

The 5 data bytes in the response have the following meaning:
[memory type] [read blocks] [read bytes per block] [write blocks] [write bytes per block]
In our example the memory type (02) is EEProm (100.000 write cycles guaranteed),
There are 4 readable blocks (0 .. 3) consisting of 8 bytes.
There are 3 writeable blocks (0 .. 2) consisting of 8 bytes.
Refer to Table 11: EEProm Memory Description for a description of the data blocks.

Request build code (header 192)

```
Command : 03 00 01 C0 3C
Response: 01 08 03 00 4C 65 76 20 20 20 4C 6F B2
```

This command returns the hopper option.
On the Evolution Hopper both low and high level sensor are standard fitted. So "Lev HiLo" will be returned.

Request address mode (header 169)

```
Command : 03 00 01 A9 53
Response: 01 01 03 00 4A B1
```

This command is used to determine how the hopper address is handled. See next table.

Bit nr	Description
B0	Address is stored in ROM
B1	Address is stored in RAM
B2	Address is stored in EEProm or battery-backed RAM
B3	Address selection via interface connector
B4	Address selection via PCB links
B5	Address selection via switch
B6	Address may be changed with serial commands (volatile)
B7	Address may be changed with serial commands (non-volatile)

Table 6: Address mode

In our example 4A is returned, meaning that the address in stored in RAM, with a selection via the interface connector and the address may be changed with serial commands.

Enter PIN number (header 218)

```
Command : 03 04 01 DA 31 32 33 34 54
Response: 01 00 03 00 FC
```

The 4 bytes (`31 32 33 34`) transmitted along with the command are the PIN numbers.
Here PIN number "`1234`" is transmitted. Correct pin codes are immediately acked.
Wrong pin codes are delayed (235 ms) acked, causing a receive timeout at the host.
A factory fresh hopper will have the pin code mechanism disabled (check with Test Hopper command).
To activate the PIN mechanism, send an ENTER NEW PIN CODE command:

Enter New PIN number (header 219)

```
Command : 03 04 01 DB 31 32 33 34 53
Response: 01 00 03 00 FC
```

The 4 bytes (`31 32 33 34`) transmitted along with the command are the new PIN numbers.
In our example "`1234`" is entered.
⇒ Once a PIN code is set, it cannot be changed or disabled.
To save the PIN code you can for example scramble it and store it in Block 0 of the EEProm.

Modify variable settings (header 165)

```
Command :03 04 01 A5 14 00 64 01 DA
Response: 01 00 03 00 FC
```

The 4 bytes (`14 00 64 01`) transmitted with the command have the following meaning:
<current limit, motor stop delay, payout timeout, single coin mode>

In this example the hopper is set in single coin mode.
(0 = multi coin mode (default), 1 = single coin mode)
In single coin mode, only 1 coin at a time can be dispensed.
⇒ This mode can only be set to multi coin mode again by resetting the hopper.
A hopper reset will set all variable settings to their default values.

Request variable settings (header 247)

```
Command : 03 00 01 F7 05
Response: 01 06 03 00 22 00 1E 29 62 00 2B
```

The 6 data bytes in the response (`22 00 1E 29 62 00`) have the following meaning:

Item	Value N		Scaling and Units	Physical value	Default value
	Hex	Dec			
Motor Current Limit	2D	45	N / 15.1 Amp	3.0 A	3.0 A
Motor Stop Delay	0	0	N ms	0 ms	0 ms
Payout Timeout	1E	30	N * 0.333 sec	10 sec	10 sec
Peak current measured	27	39	N / 15.1 Amp	2.6 A	
Supply voltage	62	98	0.2 + N * 0.127 V	12.6 V	
Connector address	0	0	N + 3	3	3

Table 7: Hopper Variable Settings

[Motor current limit]
If the current through the motor is above the threshold level (3.3 A) during 160 ms, the motor will reverse for 250 ms to clear the blocking.

[Motor stop delay]
This is the time delay after the last coin is paid out before stopping. This should ensure a clean coin exit.

[Payout Timeout]
This is the total time each coin is allowed to leave the hopper, including some reverse time in jam situations. If the hopper is empty, the motor will stop after 10 sec (default value).

[Peak current measured]
This is the maximum motor current measured, and gives you an idea about the peak current your power supply must handle. Start and stop currents in the Evo hopper are software controlled and do not have steep slopes (about 0.5 Amp/ms). If the peak current exceeds the Absolute Maximum Current Level (6.3 Amp) after a delay of 2 seconds then the hopper will return a NAK response on a Start Payout command, because the bit B0 in the Hopper Status Reg1 will be set.
A Reset command will clear the error.

[Supply voltage]
This is the measured supply voltage the hopper runs on.

[Connector address]
The address of the hopper after a power up or reset is equal to this address + 3

6.7 Hopper Dispense coins procedure

Dispensing coins using a ccTalk hopper requires some extra command steps before the actual dispense command can be executed successfully.

Test Hopper (header 163)

```
Command : 03 00 01 A3 59
Response: 01 02 03 00 C0 80 BA
```

First of all, check if any error flags are set in the status bytes.
This can be checked by sending a hopper TEST command. Two status bytes are returned in response. See Table 8: Hopper Status Register 1 and Table 9: Hopper Status Register 2.

Hopper Status Register 1		
Bit nr	**Description**	**Default value after power up**
B0	1 = Absolute maximum current exceeded	0
B1	1 = Payout timeout occurred	0
B2	1 = Motor reversed during last payout to clear a jam	0
B3	1 = Opto fraud attempt, path blocked during idle	0
B4	1 = Opto fraud attempt, short-circuit during idle	0
B5	1 = Opto blocked permanently during payout	0
B6	1 = Power-up detected	1
B7	1 = Payout disabled	1

Table 8: Hopper Status Register 1

Hopper Status Register 2		
Bit nr	**Description**	**Default value after power up**
B0	1 = Opto fraud attempt, short-circuit during payout	0
B1	1 = Single coin payout mode	0
B2	1 = Checksum A error	0
B3	1 = Checksum B error	0
B4	1 = Checksum C error	0
B5	1 = Checksum D error	0
B6	1 = Power fail during NV Memory write	0
B7	1 = Pin number mechanism enabled	0 or 1

Table 9: Hopper Status Register 2

If any error flags are set, solve the problem by inspecting the hopper and issue a RESET command.
If the hopper is build in the machine, inspecting may be difficult. There may be a coin stuck in the coin exit port, due to a heavy jam followed by a reset, or due to a power failure during a payout. The "Opto fraud attempt, path blocked during idle" flag is then set. It can be cleared again with a Reset command but is set again 333ms after the Reset command, due to the opto-test during idle (done 3x per second). If an Opto error flag is set, no payout can be started. However, if a Dispense command is transmitted within 333ms after a Reset command, the opto-flags will be cleared and the hopper may be started again to start a new payout.

In our example the first status byte (C0) indicates that a power up is detected and the hopper is disabled. The second byte (80) tells us that the PIN number mechanism is enabled. To unlock the hopper, a pin code must be send to the hopper:
If no opto-error flags are set, the hopper enable command may be issued.
After a power up or a reset the hopper is disabled. Check the status bytes again. The first data byte contains the status of the hopper. If bit B7 is set, payout is disabled.

After checking the hopper status bytes, check if any residual (uncompleted) payout is pending. This can occur if a previous payout was aborted due to a power failure, a coin jam or hopper ran out of coins. Transmit the REQUEST HOPPER STATUS command to check the status:

Request hopper status (header 166)

```
Command : 03 00 01 A6 56
Response: 01 04 03 00 00 00 01 00 F7
```

The 4 data bytes (`00 00 01 00`) in the response have the following meaning:
[event counter] [payout coins remaining] [last payout: coins paid] [last payout: coins unpaid]

[event counter]
After each valid (no communication errors) Dispense coins command, this counter is incremented. Only after a hopper reset it is set to 0. This counter should be checked each time a dispense command is transmitted, to check if the command has been received by the hopper. This should prevent sending too many or too less payout commands resulting in wrong payouts. If the hopper status event counter in incremented, then check the payout results by checking the coin counters.

[payout coins remaining]
After receiving a hopper dispense command, this counter is set with the number of coins to pay.
Each time a coin is paid, this counter is decremented.
If the payout operation completes successfully or abnormally, this counter will be set to 0.
The Host software should always check this counter if it has become 0. If it has become 0 and the coins unpaid counter is non-zero, then the dispense procedure has been aborted before all coins were dispensed. Check with the Hopper Test command if the hopper has timed-out (due to jams or empty).

[last payout: coins paid]
After receiving a hopper dispense command, this counter is set to 0.
Each time a coin is paid, this counter is incremented.
If the payout operation completes successfully, this counter will be equal to the number of coins paid since the last payout.

[last payout: coins unpaid]
This counter holds the number of coins that failed to payout after the hopper aborted the payout operation. Since the [payout coins remaining] counter is set to 0 after abnormal termination, this counter will hold the number of coins unpaid. During a payout, this counter will be set to 0.
CoinsUnpaid is only saved if the power is lost during a payout (abnormal termination).
If the hopper stops due to a payout timeout or emergency stop (normal terminations), then CoinsUnpaid is cleared. The host machine is responsible for remembering the nr coins unpaid.

Emergency Stop (header 172)

```
Command  : 03 00 01 AC 50
Response : 01 01 03 00 00 FB
```

The data byte in the response holds the number of unpaid coins since the dispensing was aborted by the Emergency Stop command. Store this result in the machine's non-volatile memory for use after power recovery.
If the hopper is running and a power failure occurs, this command can be used to stop the hopper motor and save the hopper status in case of a power failure.

⇒ Sending this command during a power down may be difficult to implement. Ensure that the ccTalk interface is still operating during a power down and that there is enough communication time to send the command. If this is not possible or impractical, do not use the command.

If the hopper was running during a power failure and the emergency stop command could not be issued, then the hopper will stop and save it's status as soon as the power dips below 8V during 20ms.

The time to save all coin counters in EEProm memory may take up to 80ms. If a coin is ejected during this EEProm update moment (possible, because the coin can already be in the coin exit port if the power fails), then this coin will not be saved in EEprom. The result will be an overpay of 1 coin. Therefore, if possible, use the Emergency Stop command to stop the hopper during a power failure, before the hopper stops due to it's own power failure detection mechanism.

In order to be sure that no payout fraud will be possible by unplugging the power of the machine during a payout, it may be wise to decrement the number of coins to be paid by 1 after power recovery.

Empty Hopper (header 25)

```
Command : 03 04 01 19 FF FF FF 00 E2
Response: 01 01 03 00 03 F8
```

Before the hopper Empty command is transmitted, be sure that the hopper is still enabled (Check with hopper Test command). If not, send hopper Enable command first. The procedure for emptying the hopper is the same as for the Dispense command. Only the command header is different (0x19).
The 3 bytes after the command (0x19) is the hopper's serial number, followed by the number of coins to pay out (dummy, set to 0).

Use the REQUEST HOPPER STATUS command to check the status of the hopper during emptying. Note that nrRemaining coins remains 1 during emptying, and nrCoinsPaid increases with each coin paid.

Enable Hopper (header 164)

```
Command : 03 01 01 A4 A5 B2
Response: 01 00 03 00 FC
```

Transmit the ENABLE HOPPER command with data byte 1 set to 165 (= A5 Hex) to enable payout. The hopper can be disabled by sending the ENABLE HOPPER command with data byte 1 set to any value other then 165. After transmitting the ENABLE HOPPER command the TEST HOPPER command can be issued again to check if the payout is enabled.
⇒ The hopper remains enabled until the hopper is reset or is disabled by command.

Once the hopper is enabled (and not blocked by a pin code) the payout can be started.
- If the hopper Product Code is "SUH1", then the DISPENSE COINS command needs an 8-byte security code from the hopper in order to start a payout. Get the security code from the hopper by transmitting the REQUEST CIPHER KEY command. Optionally the PUMP RNG command may be transmitted prior to the Request Cipher Key command to randomize the security code from the hopper even more.
- If the hopper Product Code is "SUH1-NOENCRYPT", then the DISPENSE COINS command still needs an 8-byte code, but the value of the code does not matter.
- If the hopper Product Code is "SUH1-USE_SERNR", then the DISPENSE COINS command needs it's 3-byte serial number sent along with the command to enable a payout.

Pump RNG (header 161)

```
Command : 03 08 01 A1 5B DA AA 6F 9A 5D C5 06 43
Response: 01 00 03 00 FC
```

The 8 bytes transmitted with the Pump RNG command are random numbers generated by the host. These random numbers are used by the hopper to generate a random cipher code.

Request Cipher Key (header 160)

```
Command : 03 00 01 A0 5C
Response: 01 08 03 00 69 EE 8F 1C 25 FA AB 08 20
```

The Request Cipher Key returns 8 security bytes: (`69 EE 8F 1C 25 FA AB 08`).
Once the security key from the hopper is received, the payout can be started by transmitting the DISPENSE HOPPER COINS command together with the (encrypted) security bytes and the number of coins to payout.

⇒ It is possible to disable the security bytes and replace the security bytes with the 3-byte serial number of the hopper. The hopper can be set in this mode using a special Setup program from the factory. Refer to the Product Code to check your version of the hopper. See also Request Product Code (header 244). Remember that when the unencrypted serial number is used as a dispense key, the Request Cipher Key must also be requested before the dispense command can be issued.

Dispense Hopper coins (header 167)

```
Command : 03 09 01 A7 28 DB 6A 16 7C 92 B9 C6 03 39
Response: 01 01 03 00 02 F9
```

The Cipher Key from the previous command is sent encrypted (bytes: `28 DB 6A 16 7C 92 B9 C6`), together with the dispense coin command (`A7`)and the nr coins to pay (`03`).
The response is a ACK message with 1 data byte: [event counter].
Each time a dispense command in transmitted without any communication errors, the event counter is incremented. The host software can check this value if any dispense commands are missing (due to communication errors).

If the hopper Product Code is "SUH1-USE_SERNR", the dispense command would look like follows:

```
Command : 03 04 01 A7 DF D7 0A 01 90
Response: 01 01 03 00 01 FA
```

The 3-byte serial nr (`DF D7 0A`) in the command string is followed by the nr of coins to pay (`01`).
The format of the serial number is explained in Request Serial Nr (header 242).

A NAK response is returned in the following situations (check with HOPPER TEST command):
- The coin exit is blocked
- The hopper is not enabled
- PIN code not transmitted
- Cipher key not requested
- Nr coins is not 1 in single coin mode
- Absolute Maximum Current Level has been exceeded.
- A Hopper Dispense command was sent when the hopper was already dispensing.

When the payout is started, all hopper status counters are updated:
- [event counter] is incremented (also when a NAK is received)
- [payout coins remaining] is set to nr coins to pay
- [coins paid] is set to 0
- [coins unpaid] is set to 0

During payout the counters are updated as follows:
- [payout coins remaining] is decremented each time a coin is paid
- [coins paid] is incremented a coin is paid
- [coins unpaid] is set to 0

After the payout operation has stopped normally or abnormally, the counters are updated as follows:
- [coins unpaid] is set to [payout coins remaining], except if a Emergency stop occurred due to power being lost. In this case coins unpaid is set to 0, and the host machine should store the number of coins still unpaid (which is returned by the Emergency stop command).
- [payout coins remaining] is set to 0.

This counter should always be checked during the coin dispensing. If it reached 0, the host software should check if the payment is completed or aborted by checking the coins paid and coins unpaid counters.

If a power reset occurred during a payout, all counters are saved in EEProm. These values can be used to finish the pending payout if the power is back again.

During a payout, the coin counters can be retrieved using the REQUEST HOPPER STATUS command. It is recommended to poll the hopper status each 100 ms during payout. Any display of the counter values can be monitored real-time.
If the host receives no reply to the REQUEST HOPPER STATUS command, it will be retransmitted 50ms later again.

Request Hopper Status (header 166)

```
Command : 03 00 01 A6 56
Response: 01 04 03 00 02 00 00 03 F3
```

The 4 bytes in the response (`02 00 00 03`) have the following meaning:
[event counter] [payout coins remaining] [last payout: coins paid] [last payout: coins unpaid]
In our example 2 dispense commands have been issued, 0 coins are remaining, 0 coins are paid and 3 coins are unpaid.

A final step in the payout procedure is:

Verify payout
After the dispensing of coins, the TEST command should be issued to verify if no abnormal situations have occurred. (opto blocks, jams, etc). The REQUEST HOPPER STATUS and REQUEST HOPPER DISPENSE COUNT commands should be issued to check if all coins have been dispensed properly.

- Check the dispense event counter to check if the dispense command is received.
- Check if any errors occurred during the payout with the hopper test command.
- Check if all coin counters balance with the request hopper status command.
- Check the coin level status (empty/full) (see next command)

Request Payout high/low status (header 217)

```
Command : 03 00 01 D9 23
Response: 01 01 03 00 10 EB
```

The returned data byte is the level status.
The bits of level status have the following meaning:

Bit nr	Description
0	Low Level status (1 = low level)
1	High Level status (1 = high level)
2	not used
3	not used
4	Low Level sensor is fitted
5	High Level sensor is fitted
6	not used
7	not used

Table 10: LevelStatus bit definition

In our example the low level sensor in fitted and the level status is normal.

Request Hopper dispense count (header 168)

```
Command : 03 00 01 A8 54
Response: 01 03 03 00 2E 02 00 C9
```

The 3 data bytes (`2E 02 00`) represent the 3 byte total dispense counter: 558 (LSB first).
This counter in incremented each time a coin is paid, and is not reset before any new payout.
The counter can only be reset by writing 4 zero's to the hopper dispense counter which resides in block 2 in EEProm.

The Hopper dispense life counter is the same counter has the hopper dispense counter, and resides in block 3 in EEProm. However this block can't be written to and thus the counter can't be reset to 0.

All non-volatile coin counters as well as other EEProm data can be retrieved with the Read data block command: Refer to Table 11: EEProm Memory Description.

Read Data Block (header 215)

```
Command : 03 01 01 D7 02 22
Response: 01 08 03 00 2E 02 00 D0 01 FF 00 00 F4
```

In this example Read Data Block 2 is requested.
The 8 bytes in the response (`2E 02 00 D0 01 FF 00 00`) contain the hopper dispense counter (`2E 02 00`) and a checksum(`D0`) over the counter.

Write Data Block (header 214)

```
Command : 03 09 01 D6 02 00 00 00 00 00 00 00 00 1B
Response: 01 00 03 00 FC
```

In this example 8 bytes (`00 00 00 00 00 00 00 00`) were send to EEProm Block 2.
This means that Hopper Dispense count, coins paid and coins unpaid are all reset to 0.

Block Nr.	Length (bytes)	Description	Read/Write Permission
0	8	User data	R / W
1	6	Coin name	R / W
1	2	User data	R / W
2	3	Hopper dispense count	R / W
2	1	Checksum A	R / W
2	1	Last payout: coins paid	R / W
2	1	Checksum B	R / W
2	1	Last payout: coins unpaid	R / W
2	1	Checksum C	R / W
3	3	Hopper life dispense count	R
3	1	Checksum D	R
3	1	Black Box Recorder A	R
3	1	Black Box Recorder B	R
3	1	Black Box Recorder C	R
3	1	Black Box Recorder D	R

Table 11: EEProm Memory Description

Summary dispense coins example:

1. Check the hopper status by sending the TEST command:
`Command : 03 00 01 A3 59`
`Response: 01 02 03 00 80 80 FA`
The hopper is disabled and the pin code mechanism is enabled.

2. If the pin code mechanism is enabled and the pin code is not yet transmitted, transmit the pin-code.

3. Enable the hopper by sending the ENABLE command:
`Command : 03 01 01 A4 A5 B2`
`Response: 01 00 03 00 FC`
Optionally a new TEST command can be issued to verify the status of the hopper.

Next 2 steps (4 and 5) are not necessary for the ccTalk implementation when the Serial Nr is used with the Dispense command. (Hopper Product Code is "SUH1-USE_SERNR")
4. This step is optional: Randomize hopper security code by sending PUMP RNG command:
`Command : 03 08 01 A1 5B DA AA 6F 9A 5D C5 06 43`
`Response: 01 00 03 00 FC`
The security code in the hopper is randomized using 8 random bytes.

5. This step is optional: Request the security bytes from the hopper by sending the REQUEST CIPHER KEY command:
`Command : 03 00 01 A0 5C`
`Response: 01 08 03 00 B9 FE 5F AC 75 0A 7B 98 A0`
The following security bytes are received from the hopper: `B9 FE 5F AC 75 0A 7B 98`

6. If the hopper Product Code is "SUH1", then start the payout of 3 coins by sending the DISPENSE COINS command together with the security codes:
`Command : 03 09 01 A7 63 D9 2A 95 BA D4 35 82 03 09`
`Response: 01 01 03 00 01 FA`
The host sends the (encrypted) security code `63 D9 2A 95 BA D4 35 82` together with the number of coins to pay (`03`).

If the hopper Product Code is "SUH1-USE_SERNR", the dispense command would look like follows:
`Command : 03 04 01 A7 DF D7 0A 01 90`
`Response: 01 01 03 00 01 FA`
The 3-byte serial nr (`DF D7 0A`) in the command string is followed by the nr of coins to pay (`01`).
The format of the serial number is explained in Request Serial Nr (header 242).

The data byte in the response is an event counter. Each time the DISPENSE COINS command is issued, the event counter is incremented.

7. During the payout, the number of remaining coins can be checked with the REQUEST HOPPER STATUS command:
`Command : 03 00 01 A6 56`
`Response: 01 04 03 00 02 00 03 00 F3`
From the response can be seen that 2 dispense commands have been sent, 0 coins are remaining, 3 coins have been paid and 0 coins are unpaid. Check the Hopper Status each 200ms during payout. Once [Nr Coins Remaining] is 0, the polling may be stopped and the payout can be verified.

8. Check the hopper status by sending the TEST command:
`Command : 03 00 01 A3 59`
`Response: 01 02 03 00 00 80 7A`
From the status bytes can be seen that no payout timeout, opto blockings or coin jams have occurred.

9. Optionally the payout can be verified by checking the coins paid, coins unpaid and total dispense counters using the REQUEST HOPPER STATUS (header 166) and REQUEST HOPPER DISPENSE COUNT (header 168) commands.

Host program example (only used for protocol explanation) in pseudo code

```
/* Initialize communication */
For each Hopper with HOPPER_ADDRESSx in machine do
{
    if ( SimplePoll(HOPPER_ADDRESSx) ) != ACK)
    {
         // resolve any device address problems
         While ( AddressClash(HOPPER_ADDRESSx) )  // more devices share HOPPER_ADDRESSx ?
         {
             RandomizeAddresses(0);           // give each device a new random address
             DeviceList = AddessPoll(0);      // store all received address in list

             While (DeviceList not empty)
             {
                 address = GetNextAddressFromDeviceList();
                 RequestEquipmentCategory(address);       // check device type
                 ChangeAddress(address, DEVICE_ADDRESS);  // change address to DEVICE_ADDRESS
             }
        }
        // SimplePoll(HOPPER_ADDRESSx) returned ACK. Communication Ok!

        if (RequestEquipmentCategory(HOPPER_ADDRESSx) != "Payout")
        {
             ShowMessage("DEVICE IS NOT A HOPPER!");
        }
        else
        {
            Hopper[x].PhysicalAddress = RequestVariableSettings(PHYSICAL_ADDR,
                                                                HOPPER_ADDRESSx);
             Hopper[x].SerialNr = RequestVariableSettings(SER_NR, HOPPER_ADDRESSx);
         // Optional the following items may be retrieved
             Hopper[x].ManufacturerID = ...
             Hopper[x].ProdCode = ...
             Hopper[x].SoftwareRev = ...
             Hopper[x].CommsRev = ...
             Hopper[x].HopperCoin = ...
             Hopper[x].BuildCode = ...
         }
     }


// Product Configuration details


// Check for pending Payout after a power fail recovery
```

```
// Dispense Coins from HOPPER_ADDRESSx
HopperAddress = HOPPER_ADDRESSx;
HopperType = RequestProductCode(HopperAddress);       // get hopper type
HopperStatus = TestHopper(HopperAddress);             // get hopper status bytes

if (HopperStatus & PAYOUT_ERROR_FLAGS) == 0)  // no opto-errors, max current exceeded, etc?
{
      if (HopperStatus & RESET_OCCURRED)      // after a reset, send pin code if necessary
     {
            // Pin Number Unlocking
            if (HopperStatus & PIN_NUMBER_REQUIRED)
            {
             SetPinNumber(PinCode, HopperAddress);// unlock hopper (required a after reset)
       }
     }

    if (HopperStatus & HOPPER_DISABLED)
    {
         EnableHopper(HopperAddress);                 // enable hopper
    }

     SecurityBytes = RequestCipherKey(HopperAddress);  // ReqCipherKey, not needed for SCH2-
                                                       // USE_SERNR
     if (HopperType == "SUH1" )       // use security bytes?
     {
         DispenseCoins(HopperAddress, SecurityBytes, NrCoinsToPay);
     }
    else if (HopperType == "SUH1-NOENCRYPT")  // use dummy bytes? (security disabled)
    {
         DispenseCoins(HopperAddress, DummyBytes, NrCoinsToPay);
    }
    else if (HopperType == "SUH1-USE_SERNR")  // use serial nr as dispense key?
    {
       DispenseCoins(HopperAddress, SerialNr, NrCoinsToPay); // serial number already
                                                              requested during
                                                              initialization
    }
}
else// hopper errors
{
     ResetHopper();       // send reset command to clear error flags, and try again
     // in case of opto-errors, send dispense command within 333 ms after reset command
}

// Check hopper status during payout
do
{
     // request hopper status each 100ms for real-time display of count values
    HopperCounters = RequestHopperStatus(HopperAddress);  // update status counters
}
while (HopperCounters.NrCoinsRemaining > 0)
```

```
// Verify Dispense procedure (may be extended of coarse)
HopperStatus = TestHopper(HopperAddress);         // get hopper status bytes
if (HopperCounters.NrUnPaidCoins > 0)             // payout completed ?
{
    if (HopperStatus & PAYOUT_TIMEOUT_OCCURED)        // no, hopper timeout ?
    {
        ShowMessage("Hopper Timeout occurred");
    }

   if (HopperStatus & JAM_OCCURED)       // hopper jammed during payout ?
   {
        ShowMessage("Hopper probably jammed");
   }
}

// Check Hopper Levels
HopperLevel = RequestHopperCoinLevel(HopperAddress);
if (HopperLevel & LOW_LEVEL)
{
   ShowMessage("Hopper nearly empty");
}

// Check Total Dispense counter, etc
```

6.8 Coin Jams during payout

During a payout, a coin may block the hopper (for example if the hopper is loaded too heavily, or a wrong coin has slipped into the hopper). In this case the motor current will rise quickly. A fully blocked hopper motor will draw peak currents up to 7 Amp. When the power supply is not able to deliver this peak current, the voltage on the hopper will drop.

We recommend the following power supply types from Suzo:

- 42PP0530 (dual supply: 5VDC and 24VDC)

When the voltage drops below the POWER_FAIL_TRESHOLD (see Table 1) during 20ms, the hopper will reset, aborting the payout procedure. This 20ms may seem a short time, but remember that if the power supply is 18.0V and falling, the hopper must be able to stop and save all payout settings before the power is gone totally.
As long as the power supply on the hopper is above the POWER_FAIL_TRESHOLD level, the hopper will start an anti-jam operation by reversing and restarting the hopper motor.

If the motor current is greater than 3.0A during 160ms, the motor will start reversing in order to un-jam the hopper. During anti-jamming, the current my rise to levels of 4 Amp peak.

6.9 Power failures

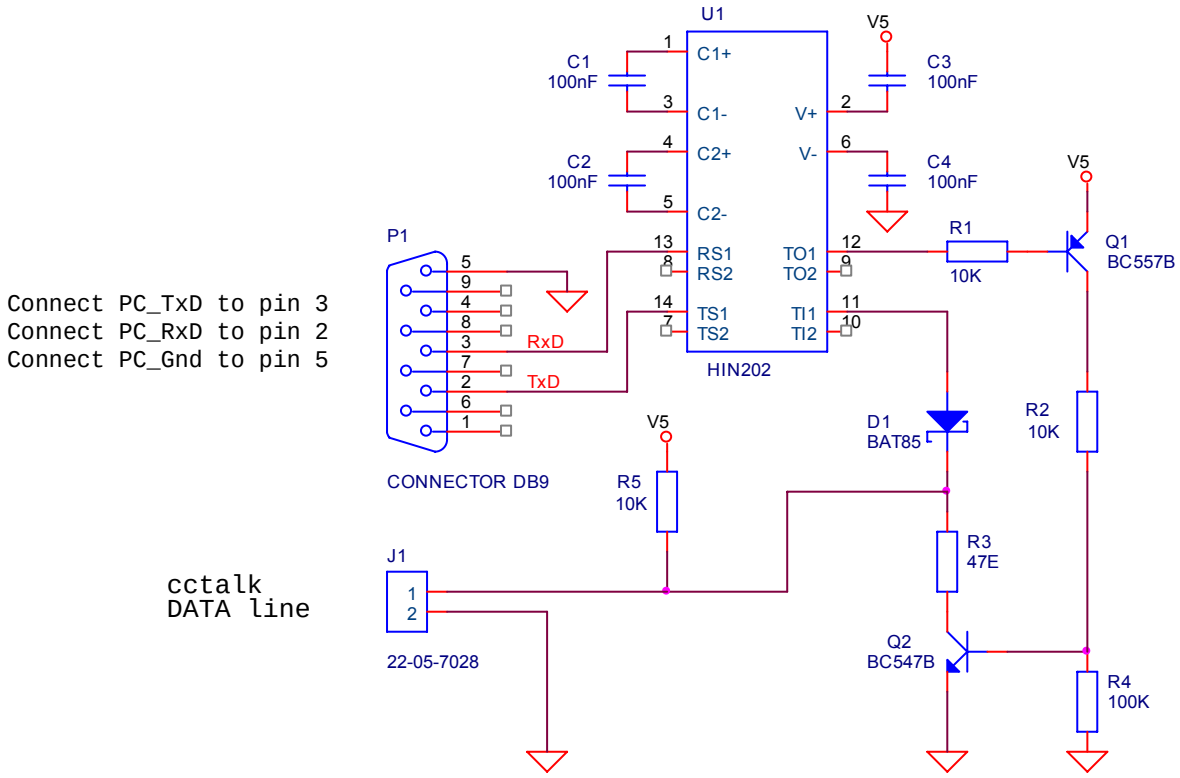
The hopper measures the voltage each ms.
If the voltage drops below the POWER_FAIL_TRESHOLD during 20ms, the hopper will stop immediately if it was running, and save all counters (CoinsPaid, CoinsUnpaid) in Non-volatile memory.
A power dip of more then 20ms will stop the hopper and save all data.
If the power returns again, the host software must retrieve the hopper status and take appropriate action if there are, for example, unpaid coins left.

If the host machine has early-power down notification, the host machine may send an 'Emergency Stop' command to the hopper so that the hopper will stop and save all data.
The NrCoinsRemaining is transmitted back to the host. The host machine must ensure that this value is stored in it's own non-volatile memory.
See also Emergency Stop (header 172).

7. Hopper Application

7.1 PC Interface Circuit

The circuit diagram below shows an interface that can be used to connect a 9-pin serial port of a PC to a ccTalk data bus.



8. Technical Specifications

8.1 Coin Sizes

Track type	Coin sizes	Color	Art. nr.
Standard (€2, €1, €0.50, €0.20)	21.01 – 30.00 mm x 1.25 – 3.30 mm	Red	EV0050
Euro (€2, €1, €0.50, €0.20, €0.10, €0.05)	19.00 – 26.40 mm x 1.50 – 2.50 mm	Yellow	EV0050-3
Euro small (€0.10, €0.05, €0.02, €0.01)	16.25 – 20.90 mm x 1.00 – 3.10 mm	Green	EV0050-4

Table 12: Coin size vs Track type

8.2 Capacity

$$\text{Capacity} = \text{Hopper volume} / \text{Coin volume} = \frac{1{,}200{,}000}{\frac{\pi \times D^2}{4} \times T}$$

D = Coin diameter (mm)
T = Coin thickness (mm)

8.3 Connector

Pin	Description
1	Power supply 0 Vdc
2	Power supply 0 Vdc
3	Coin exit output (not mandatory for ccTalk)
4	Address select 1
5	ccTalk data line
6	High or Top level sense output (not mandatory for ccTalk)
7	Low level sense output (not mandatory for ccTalk)
8	Address select 2
9	Power supply 24Vdc
10	Power supply 24Vdc
11	Coin exit output (not mandatory for ccTalk)
12	Address select 3

Figure 5: Connector pinout

8.4 Electrical Interface

Electrical Interface	ccTalk
Voltage: nominal	24 VDC
minimum	20 VDC
maximum	26 VDC
Current (typical):	
idle	40 mA
empty	0.2 A
full	0.7 A
surge	2.5 A
Vtrip (Power fail threshold)	18.0 V during 20ms
Motor_current_reverse level	> 3.0 A during 160 ms
Thermal fuse protection	5.5

Table 13: Electrical Interface

Parameter	Description	Value	Units
Brate	Serial communication speed	9600	baud
Trxout	Receive data timeout	25	ms
TPinit	Power up initialization time	< 600	ms
TRinit	Hardware reset init time	< 20	ms
TSinit	Software reset init time	< 40	ms
PWMFreq	Motor drive PWM frequency	20	kHz
TMreverse	Motor reverse time	250	ms
Ifuse	Fuse trip time @ 5.5 A	5	sec
Tlevdeb	Level sensor debounce time	2	sec
TEESave	EEProm write time per byte	7	ms
TEESave-PowerDown	Save time hopper status at power down	20 - 90	ms
TResponse	Command response time (except for command headers 253, 252)	< 2	ms

Table 14: Timing specifications

8.5 Logic Inputs

Description	Value (Vin)
Recommended logic 0 input	<= 0.6V
Recommended logic 1 input	>= 2.6V

Figure 6: Logic inputs

8.6 Logic Outputs

Description	Value
Output level 0	<= 0.3 at 50mA
Max sink current	100mA
Max pull up voltage	30Vdc
Coin exit typical pulse width	70 – 120ms

Figure 7: Logic outputs

8.7 Interface Options

- Standard Parallel (EV1000)
- ccTalk Italian version (EV2000)
- ccTalk MC version (EV3000)

ccTalk MC non encrypted version (EV4000)

8.8 Payout Rate

appr. 4 coins per sec.

8.9 EMC approval

• EN 55014-1 (2000) + A1 (2001) + A2 (2002): Electromagnetic Compatibility: Requirements for household appliances, electric tools and similar apparatus - Part 1: Emission.
• EN 55014-2 (1997) + A1 (2001): Electromagnetic Compatibility - Requirements for household appliances, electric tools and similar apparatus - Part 2: Immunity – Product family standard.
• EN 61000-3-2 (2000): Electromagnetic compatibility - Part 3-2: Limits - Limits for Harmonic current emissions (equipment input current < 16A per phase).
• EN 61000-3-3 (1995) + A1 (2001): Electromagnetic compatibility - Part 3: Limits - Section 3: Limitation of voltage fluctuations and flicker in low- voltage supply systems for equipment with rated current < 16A.

FCC approval:

• 47CFR15: Radio Frequency Devices

8.10 Environment

Description	Value
Operating temperature	0 to 60ºC
Storage temperature	-20 to 60ºC
Life	Up to 3 million coins
Mounting	±3º of vertical in any direction

Note: DO NOT use the hopper in an explosive atmosphere

9. Dimensions

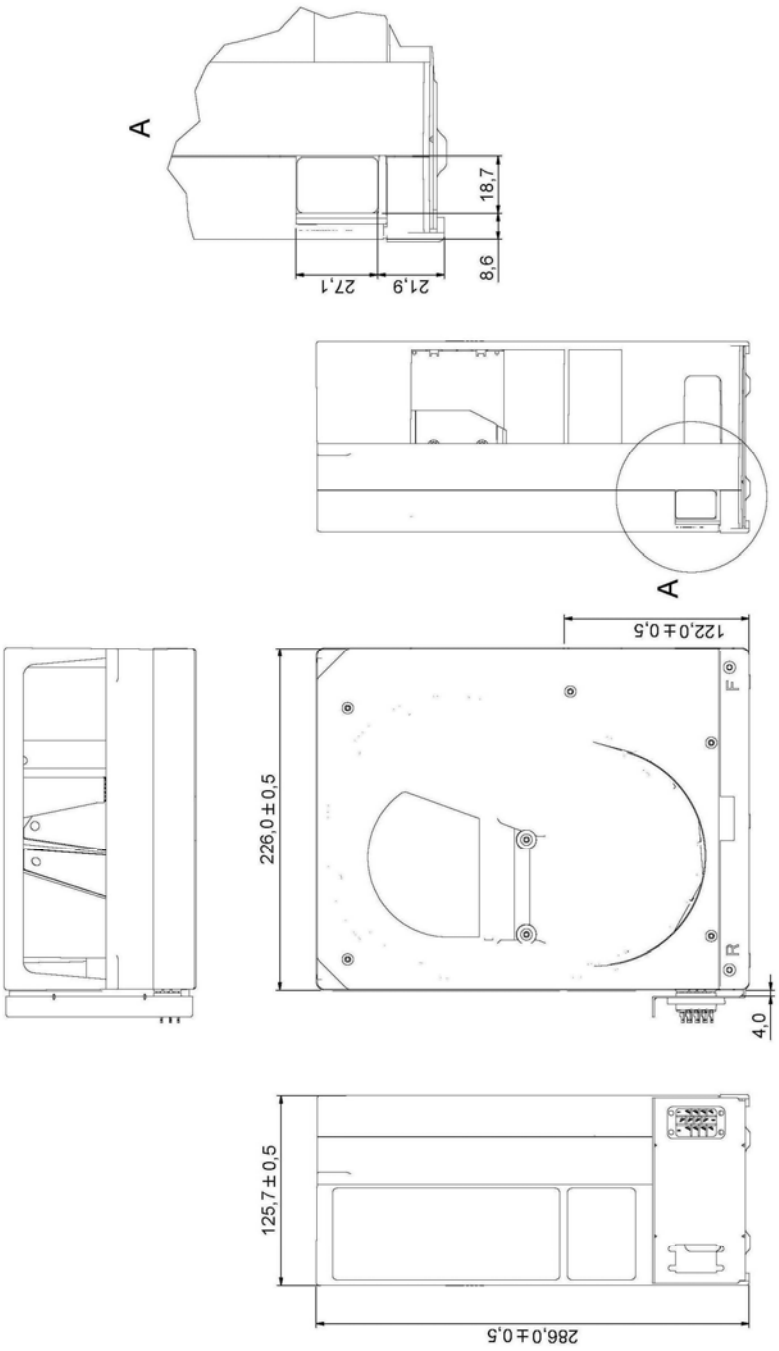


Figure 8: Hopper dimensions

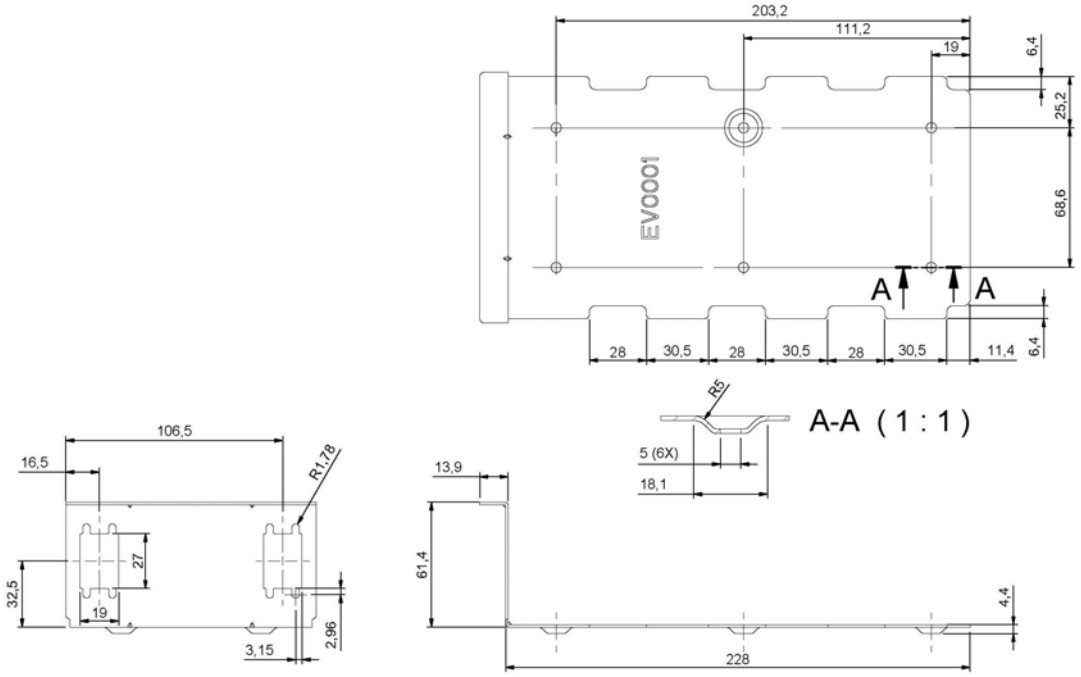
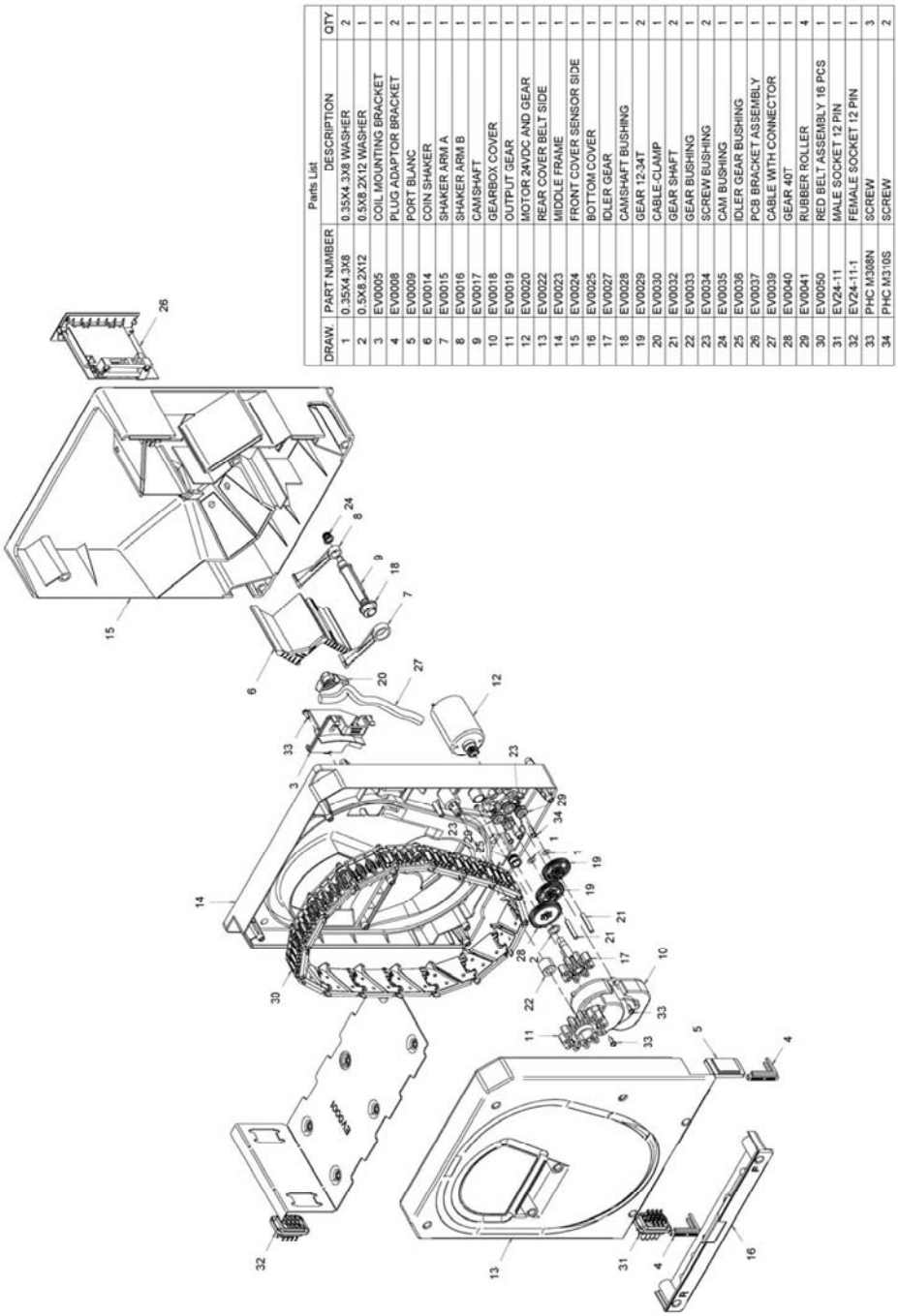


Figure 9: Base plate dimensions

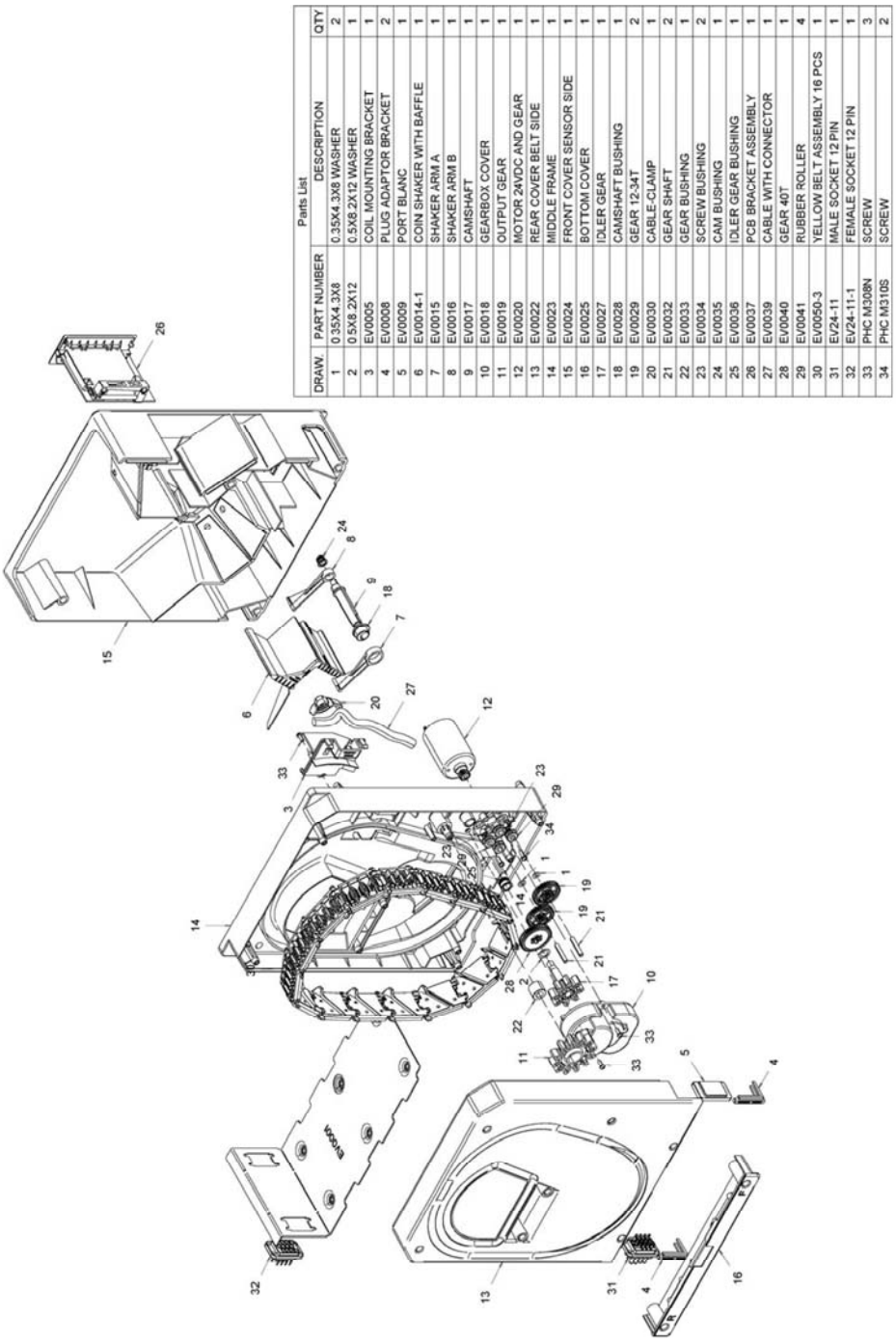
10. Exploded View



Parts List

DRAW.	PART NUMBER	DESCRIPTION	QTY
1	0.35X4.3X8	0.35X4.3X8 WASHER	2
2	0.5X8.2X12	0.5X8.2X12 WASHER	1
3	EV0005	COIL MOUNTING BRACKET	1
4	EV0008	PLUG ADAPTOR BRACKET	2
5	EV0009	PORT BLANC	1
6	EV0014	COIN SHAKER	1
7	EV0015	SHAKER ARM A	1
8	EV0016	SHAKER ARM B	1
9	EV0017	CAMSHAFT	1
10	EV0018	GEARBOX COVER	1
11	EV0019	OUTPUT GEAR	1
12	EV0020	MOTOR 24VDC AND GEAR	1
13	EV0022	REAR COVER BELT SIDE	1
14	EV0023	MIDDLE FRAME	1
15	EV0024	FRONT COVER SENSOR SIDE	1
16	EV0025	BOTTOM COVER	1
17	EV0027	IDLER GEAR	1
18	EV0028	CAMSHAFT BUSHING	1
19	EV0029	GEAR 12-34T	2
20	EV0030	CABLE-CLAMP	1
21	EV0032	GEAR SHAFT	2
22	EV0033	GEAR BUSHING	1
23	EV0034	SCREW BUSHING	2
24	EV0035	CAM BUSHING	1
25	EV0036	IDLER GEAR BUSHING	1
26	EV0037	PCB BRACKET ASSEMBLY	1
27	EV0039	CABLE WITH CONNECTOR	1
28	EV0040	GEAR 40T	1
29	EV0041	RUBBER ROLLER	4
30	EV0050	RED BELT ASSEMBLY 16 PCS	1
31	EV24-11	MALE SOCKET 12 PIN	1
32	EV24-11-1	FEMALE SOCKET 12 PIN	1
33	PHC M308N	SCREW	3
34	PHC M310S	SCREW	2

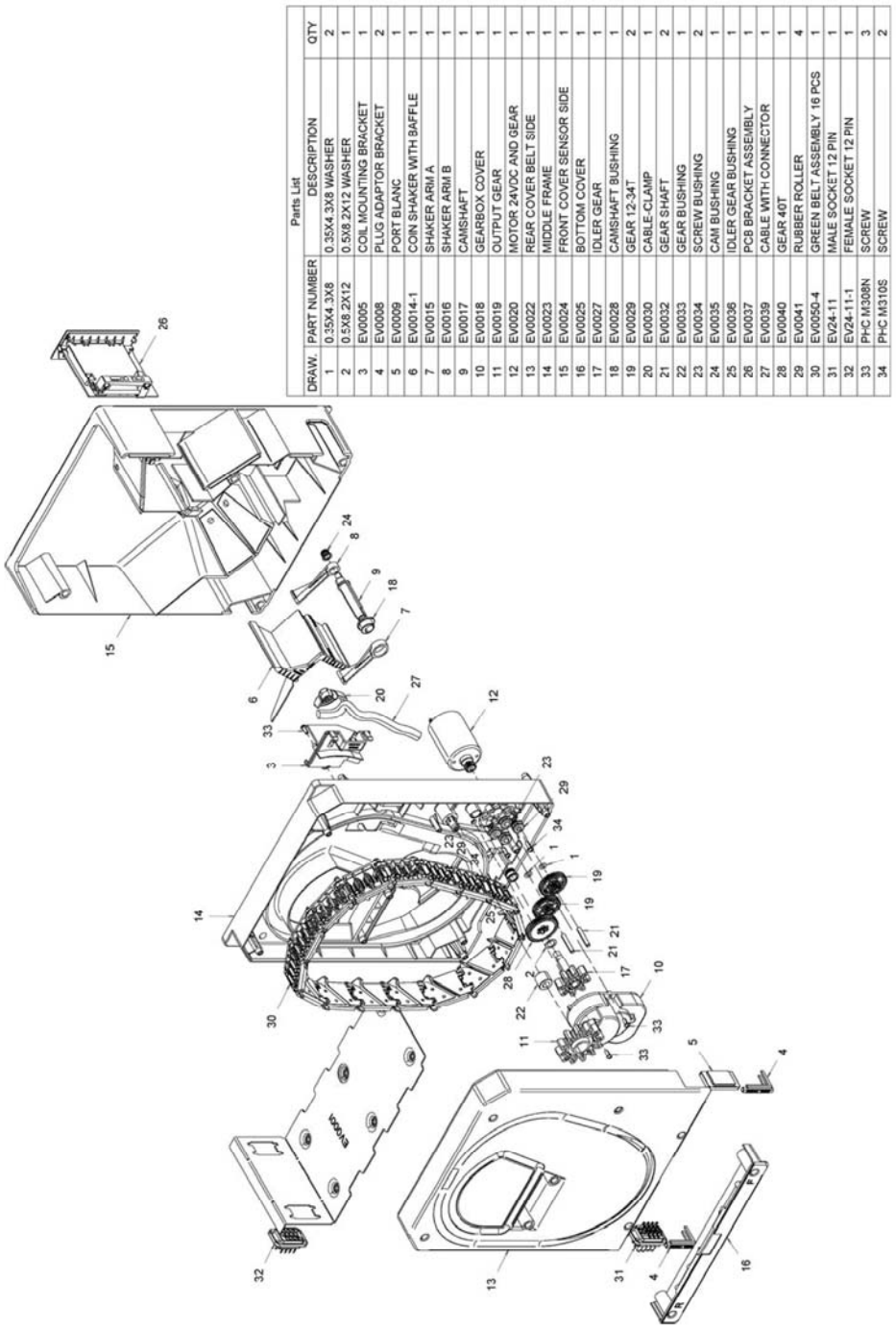
Figure 10: 21.01 – 30.00 mm series



Parts List

DRAW.	PART NUMBER	DESCRIPTION	QTY
1	0.35X4.3X8	0.35X4.3X8 WASHER	2
2	0.5X8.2X12	0.5X8.2X12 WASHER	1
3	EV0005	COIL MOUNTING BRACKET	1
4	EV0008	PLUG ADAPTOR BRACKET	2
5	EV0009	PORT BLANC	1
6	EV0014-1	COIN SHAKER WITH BAFFLE	1
7	EV0015	SHAKER ARM A	1
8	EV0016	SHAKER ARM B	1
9	EV0017	CAMSHAFT	1
10	EV0018	GEARBOX COVER	1
11	EV0019	OUTPUT GEAR	1
12	EV0020	MOTOR 24VDC AND GEAR	1
13	EV0022	REAR COVER BELT SIDE	1
14	EV0023	MIDDLE FRAME	1
15	EV0024	FRONT COVER SENSOR SIDE	1
16	EV0025	BOTTOM COVER	1
17	EV0027	IDLER GEAR	1
18	EV0028	CAMSHAFT BUSHING	1
19	EV0029	GEAR 12-34T	2
20	EV0030	CABLE-CLAMP	1
21	EV0032	GEAR SHAFT	2
22	EV0033	GEAR BUSHING	1
23	EV0034	SCREW BUSHING	2
24	EV0035	CAM BUSHING	1
25	EV0036	IDLER GEAR BUSHING	1
26	EV0037	PCB BRACKET ASSEMBLY	1
27	EV0039	CABLE WITH CONNECTOR	1
28	EV0040	GEAR 40T	1
29	EV0041	RUBBER ROLLER	4
30	EV0050-3	YELLOW BELT ASSEMBLY 16 PCS	1
31	EV24-11	MALE SOCKET 12 PIN	1
32	EV24-11-1	FEMALE SOCKET 12 PIN	1
33	PHC M3X8N	SCREW	3
34	PHC M3105	SCREW	2

Figure 11: 19.00–26.40 mm series



Parts List

DRAW.	PART NUMBER	DESCRIPTION	QTY
1	0.35X4.3X8	0.35X4.3X8 WASHER	2
2	0.5X8.2X12	0.5X8.2X12 WASHER	1
3	EV0005	COIL MOUNTING BRACKET	1
4	EV0008	PLUG ADAPTOR BRACKET	2
5	EV0009	PORT BLANC	1
6	EV0014-1	COIN SHAKER WITH BAFFLE	1
7	EV0015	SHAKER ARM A	1
8	EV0016	SHAKER ARM B	1
9	EV0017	CAMSHAFT	1
10	EV0018	GEARBOX COVER	1
11	EV0019	OUTPUT GEAR	1
12	EV0020	MOTOR 24VDC AND GEAR	1
13	EV0022	REAR COVER BELT SIDE	1
14	EV0023	MIDDLE FRAME	1
15	EV0024	FRONT COVER SENSOR SIDE	1
16	EV0025	BOTTOM COVER	1
17	EV0027	IDLER GEAR	1
18	EV0028	CAMSHAFT BUSHING	1
19	EV0029	GEAR 12-34T	2
20	EV0030	CABLE-CLAMP	1
21	EV0032	GEAR SHAFT	2
22	EV0033	GEAR BUSHING	1
23	EV0034	SCREW BUSHING	2
24	EV0035	CAM BUSHING	1
25	EV0036	IDLER GEAR BUSHING	1
26	EV0037	PCB BRACKET ASSEMBLY	1
27	EV0039	CABLE WITH CONNECTOR	1
28	EV0040	GEAR 40T	1
29	EV0041	RUBBER ROLLER	4
30	EV0050-4	GREEN BELT ASSEMBLY 16 PCS	1
31	EV24-11	MALE SOCKET 12 PIN	1
32	EV24-11-1	FEMALE SOCKET 12 PIN	1
33	PHC M308N	SCREW	3
34	PHC M310S	SCREW	2

Figure 12: 16.25 – 20.90 mm series