Flow Hopper

Model: 18-ccTalk
Product Manual
Pre release
Version 0.6 / Februari 2011

SUZOHAPP

Antonie van Leeuwenhoekstraat 9
3261 LT Oud-Beijerland
The Netherlands

Phone : +31(0)186-643333
Fax : +31(0)186-643322
Email : info@suzo.com
Web : www.suzo.com

1. Introduction

The Flow hopper is a new and innovative addition to the Suzo-Happ range of hoppers. This through hole hopper has been re-engineered and the result is an improvement over existing hoppers in it's performance. The Flow hopper is suitable for many application such as: Vending, Parking, Change, Amusement and Gaming machines. Made from extremely strong, wearing resistant plastics in combination with state of the art electronics resulting in low cost of ownership.

The Flow hopper is available in two interfaces. Standard parallel interface or ccTalk serial interface.

Supplied with standard quick-fit and release plate.

Features:

- Compatible with other popular hopper devices.
- One configuration for euro coins. 10ct to 2 euro coins.
- Disk can reach a payout speed of 7 coins per second for euro's.
- 3 different cup extensions. Small medium and large.
- Reliable through hole coin disk design.
- Clean design no external wires run for level sensors.
- Integrated fall tube for simple dual hopper cabinet construction.
- CcTalk compatible. Non encrypted and encrypted modes.
- Power supply 12-24V.
- Full bridge Mos-Fet motor drive.
- Pulse Width Modulated (20kHz) motor control giving:
    - constant pay out speed control.
    - low motor start-up current, prevents systems power-dips.
- Continuous opto sensor check.
- High sensitive opto sensor
- Anti-jam operation prevents hopper blocking.
- Low level sense are standard on the Flow hopper.
- Hi level sense are optional on the Flow hopper.
- CcTalk interface and standard 10pin connector.
- Competitively priced.

Attention!

Always turn power off before removing or installing the hopper in order to prevent any damage due to surge currents!

2. Change hopper for different coin sizes

The hopper is configured to accept most Euro coins ((€0,20 to €2.00) and the British Pound . For special situations there are other disks available adapt the hopper to other coins. Changing configuration requires removing the cup and changing two parts, the payout disk and the insert plate. For correct parts for you coin see the following two tables.

Payout discs				
Mark	Diameter (mm)	Thickness (mm)	Part no	Currency
13	16.00-18.99	1.50-2.09	18-0020-13	€0.01 €0.02
14	19.00-21.99	1.50-2.09	18-0020-14	€0.05 €0.10
15	22.00-28.49	1.50-2.09	18-0020-15	
16	28.50-32.00	1.50-2.09	18-0020-16	
23	16.00-18.99	2.10-3.20	18-0020-23	
24	19.00-21.99	2.10-3.20	18-0020-24	
25	22.00-28.49	2.10-3.20	18-0020-25	€0.20 €0.50 €1.00 €2.00 £1.00
26	28.50-32.00	2.10-3.20	18-0020-26	

Table 1: Payout discs

Coin insert plates			
Mark	Diameter (mm)	Part no	Currency
3	16.00-18.99	18-0010-3	€0.01 €0.02
4	19.00-21.99	18-0010-4	€0.05 €0.10
5	22.00-28.49	18-0010-5	€0.20 €0.50 €1.00 €2.00 £1.00
6	28.50-32.00	18-0010-6	

Table 2: Coin insert plates

2.1. Hopper disassembly

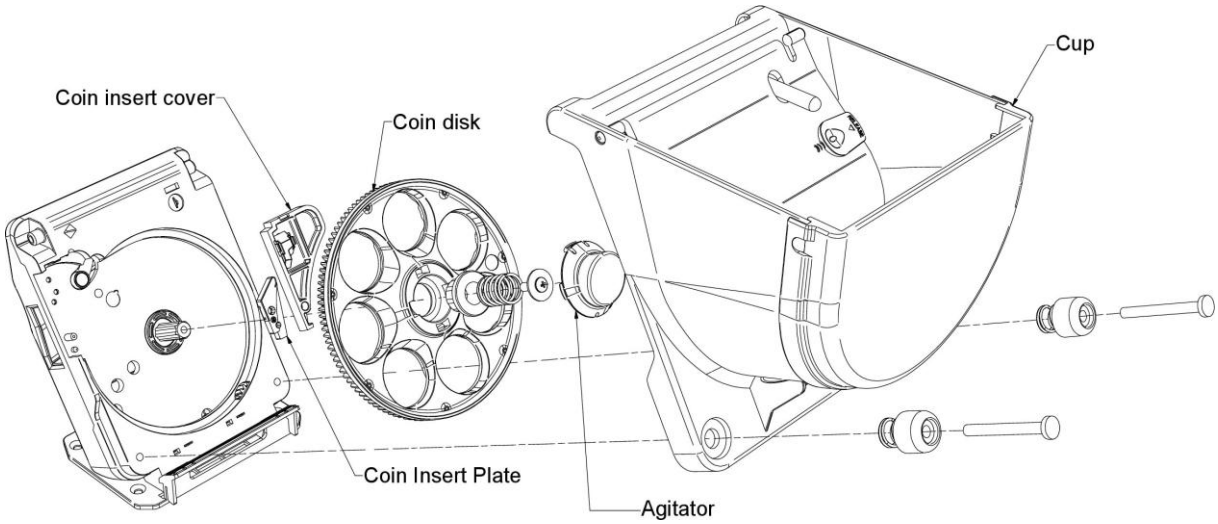


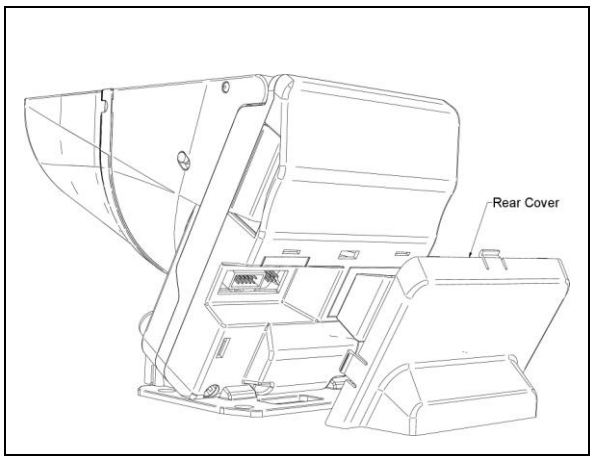
Figure 1.

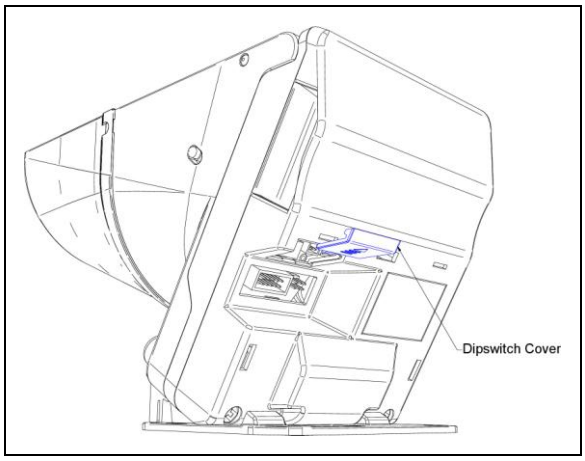
1. Remove the two screws at the bottom of the cup.
2. You can now lift the cup of the hopper.
3. Use a flat screw driver to remove the agitator.
4. You can now loosen the screw and remove the coin disk.
5. The coin insert cover can now be removed.
6. You can now switch the coin insert plate.

To reassemble place the correct coin insert plate and disk. Then follow the above steps in reverse order.

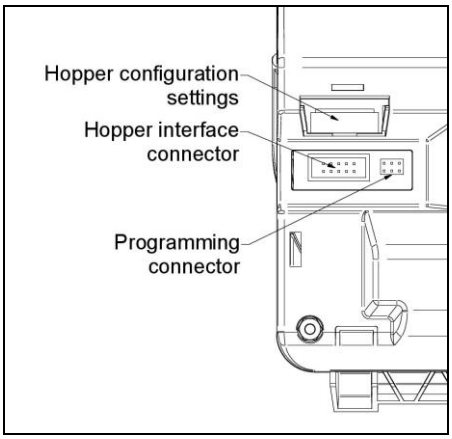
3. Protocol selection

You can select between two ccTalk protocols for compatibility with excising machines. There is a 8 way dipswitch at the rear of the hopper where the protocol can be selected. First remove the rear cover to access the connector area. Next remove the dipswitch cover.





The settings are accessible above the hopper interface connector and the programming connector.



3.1. Device Address

All Flow Hoppers leave the factory with **address 3**
The address can be changed with serial commands. Unless you have an application requiring more than one Flow 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 Flow 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.

The hopper models are equipped with a DIP-switch located inside the hopper (Hopper configuration settings). With the switches 1, 2, 3 and 4 up to 16 Hoppers.
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	Dip Switch 3	Dip Switch 4	Address
OFF	OFF	OFF	OFF	3
ON	OFF	OFF	OFF	4
OFF	ON	OFF	OFF	5
ON	ON	OFF	OFF	6
ON	ON	OFF	OFF	7
-	-	-	-	-
ON	ON	ON	ON	18

Address dipswitch setting

3.2. Setting mode

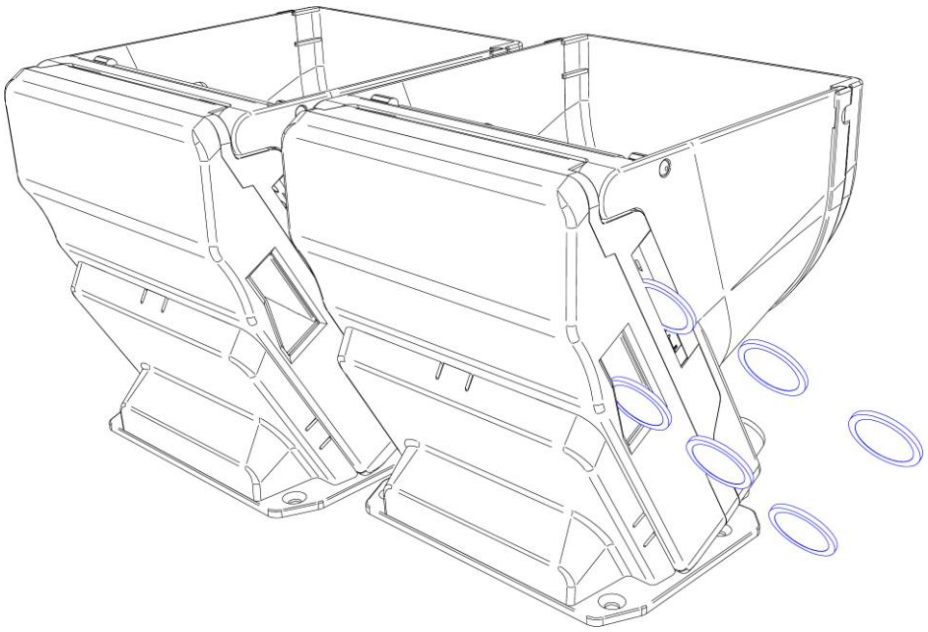
DIP-Switch 5 and 6 are not used.
Switches 7 and 8 are used to select the type of ccTalk protocol. See table below.

Mode	Dip Switch 7	Dip Switch 8
ITA serial	OFF	OFF
MC encrypted	OFF	ON

ccTalk protocol dipswitch setting

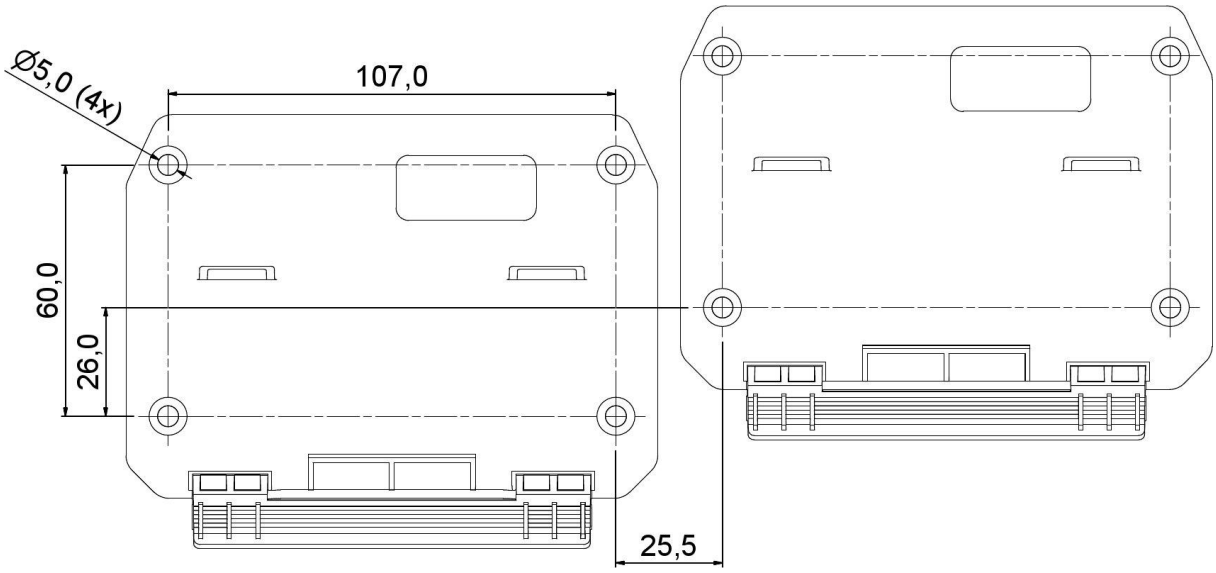
4. In-Line configuration

Two hoppers can be placed next to each other for optimal use of space inside a cabinet. The hopper has a fall tube integrated in the rear of the housing.



4.1. Platform placement

See the following figure for the optimal placement of the hopper brackets in your cabinet.



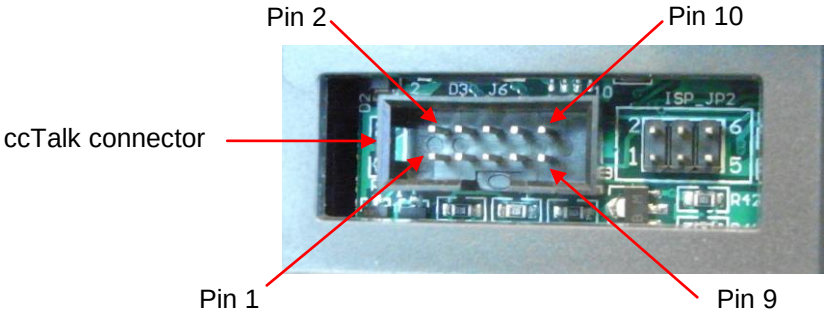
5. CcTalk implementation on Suzo Flow Hopper

9600 baud, 1 start bit, 8 data bits, no parity, 1stop bit
+12V .. +24V nominal supply
+5V data pull-up
supply sink
slave device
8-bit addition checksum

5.1. ccTalk Connector Pinout

Connector type: Molex 8624 series or similar(10 pins (2x5) of 2.5mm).
Be careful with the pin numbering of the connector, because not all manufacturers start numbering from the same pin.

Pin	Function
1	DATA
2	
3	
4	0V
5	
6	
7	+Vs
8	0V
9	
10	+Vs



Connector pinout

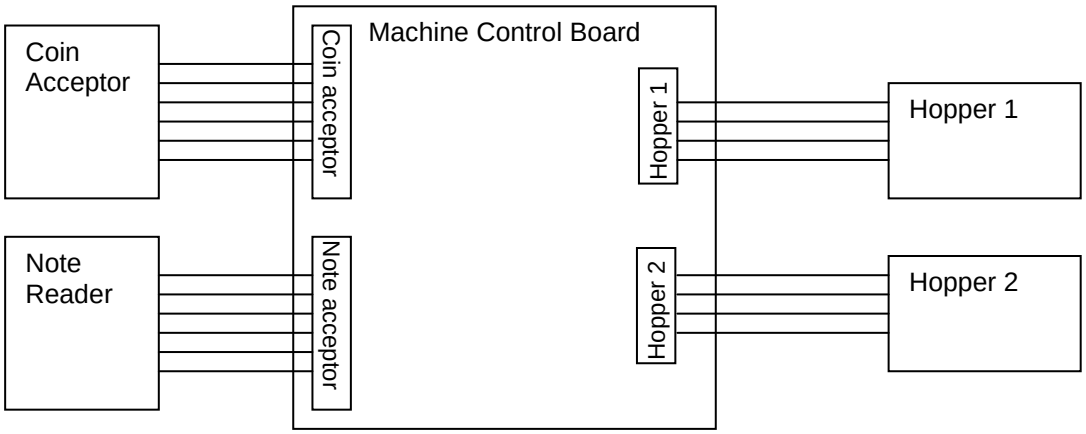
Pins 4 and 8 are linked together and pins 7 and10 are linked together, and can be used to daisy-chain the power wires from hopper to hopper.

6. 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.

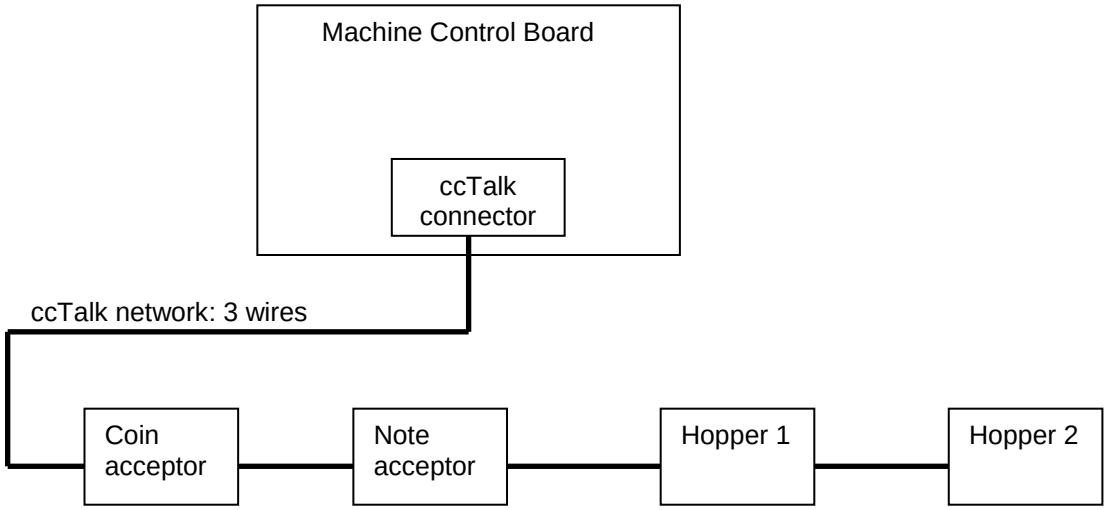
Serial Communication: One (cheap) network cable does it all

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.



Traditional machine wiring

You 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.



Network wired machine

shows how a system looks if a ccTalk network is used.
As can be, 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.

6.1. 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.

6.2. 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.

6.3. 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.

6.4. 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.

7. Commands

The following ccTalk commands are currently implemented in the ccTalk Flow hopper.

Header	Data bytes	Response (default)	Description
254	<none>	ACK	Simple poll
253	<none>	{variable delay} [slave address]	Address poll
252	<none>	{variable delay} [slave address]	Address clash
251	[new address]	ACK	Address change
250	<none>	ACK	Address random
247	<none>	[current_limit] [motor stop delay] [payout timeout] [max current measured] [supply voltage] [connector address]	Request variable set
246	<none>	“Suzo Int (NL)”	Request manufacture id
245	<none>	“Payout”	Request equipment category id
244	<none>	“Payout”	Request product code (Response depends on factory settings)
242	<none>	[serial 1- LSB] [serial 2] [serial 3 – MSB]	Request serial number
241	<none>	“Flow V1.0”	Request software version
236	<none>	[level status empty/full] (bit 0=empty detection bit1=full detection)	Read level empty/full states
219	[PIN1] [PIN2] [PIN3] [PIN4]	ACK	Enter new PIN number
218	[PIN1] [PIN2] [PIN3] [PIN4]	ACK	Enter PIN number
217	<none>	[level status] (bit 0 set: low level)	Request payout level status
215	[block number]	[data 1] [data 2] … [data 8]	Read data block
214	[block nr] [data 1] … [data 8]	ACK	Write data block
192	<none>	“Lev Lo”	Request build code
172	<none>	[payout coins remaining]	Emergency stop
171	<none>	“--------“	Request hopper coin
169	<none>	[address mode]	Request address mode
168	<none>	[nr coins 1 – LSB] [nr coins 2] [nr coins 3 – MSB]	Request hopper dispense count
167	[sec 1] [sec 2] [sec 3] [sec 4] [sec 5] [sec 6] [sec 7] [sec 8] [N coins]	[event counter]	Dispense hopper coins
166	<none>	[event counter] [payout coins remaining] [last payout: coins paid] [last payout: coins unpaid]	Request hopper status
165	[current limit] [motor stop delay] [payout TO] [max current measured] [supply voltage] [connector address]	ACK	Modify variable set
164	[enable code]	ACK	Enable hopper
163	<none>	[hopper status register1] [hopper status register 2]	Test hopper
161	[rnd 1] [rnd 2] [rnd 3] [rnd 4] [rnd 5] [rnd 6] [rnd 7] [rnd 8]	ACK	Pump RNG
160	<none>	[key 1] [key 2] [key 3] [key 4] [key 5] [key 6] [key 7] [key 8]	Request cipher key
004	<none>	[ccTalk level] [major revision] [minor revision]	Request comms revision
003	<none>	ACK	Clear comms status variables
002	<none>	[rx timeouts] [rx bytes ignored] [rx bad checksums]	Request comms status variables
001	<none>	ACK	Reset device

Table 3: Hopper commands

A detailed explanation of all commands follows in the next sections.

7.1. 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.

7.2. Hopper Setup and Initialisation commands

Before the hopper can be used, a number of initialisation 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
```

The command is sent with the destination address of 0
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). 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 06 03 00 31 33 32 2D 49 54 96
```

`The 6 bytes(31 33 32 2D 49 54)` in the response have the following meaning:
[ccTalk level] [major revision] [minor revision] [configuration]. In our example: 1 3 2 – I T (dipswitch 8 OFF, MC encrypted) and 1 3 2 - M C (dipswitch 8 ON, ITA serial).

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
```

After the hopper receives this command, the hopper takes the following actions:
- first, save all counter data in EEProm (if data has changed, this will take up to 90ms)
- then, send an ACK message to the host
- finally, perform a software reset (about 40 ms)

During software resetting (can take up to 130ms) the hopper will not respond to commands.

⇒ After sending a reset command, wait at least 100ms before generating a receive timeout (followed by a retransmission). The ACK from the hopper may be delayed for 90ms in case data has to be saved. For all other commands the receive timeout is 25 ms.
After the reset, all hopper variables (including status flags) will be reset to their default values.
⇒ A power-up reset (hardware reset) takes about 630 ms, during which the hopper will not respond to any commands.

Request Manufacturer id (header 246)

```
Command : 03 00 01 F6 06
Response: 01 09 03 00 53 75 7A 6F 2D 48 61 70 70 8C
```

The data bytes in the response (`53 75 7A 6F 2D 48 61 70 70)` give the manufacturer: `"Suzo-Happ"`. 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 06 03 00 50 61 79 6F 75 74 74
```

The data bytes in the response `(50 61 79 6F 75 74)` mean `"Payout"`, meaning the Flow Hopper.

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.
When dipswitch 8 is OFF (ITA serial), 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 0B
Response: 01 14 03 00 53 54 43 32 30 31 30 2D 56 31 2E 31 46 2D 50 61 79 6F
75 74 33
```

The response data byte (`53 54 43 32 30 31 30 2D 56 31 2E 31 46 2D 50 61 79 6F 75 74 33`) mean `"STC2010-V1.1F-PAYOUT"`.

Read level Empty/Full States (header 236)

```
Command : 03 00 01 EC 10
Response: 01 01 03 00 03 7A
```

The data bytes in the response (`03`) meaning hopper is empty.
Detect the value of the empty and full sensor. The follow table shows the value.

Load	Bit 0	Bit 1
Empty	1	1
Half Load	0	1
Full	0	0

Request build code (header 192)

```
Command : 03 00 01 C0 3C
Response: 01 06 03 00 50 61 79 6F 75 74 74
```

The data bytes in the response `(50 61 79 6F 75 74)` mean `"Payout"`.
There is only one option on the Flow hopper: this is the device assembly code.

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 4: 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.
If the pin mechanism is active (different to 0), all commands implemented in the Flow hopper, except the header 253 “Address Poll” are protected. If the PIN number is activated and the command doesn’t have the correct PIN, there will be no answer from the Flow hopper. It must be sent after every “power off” and “reset”. 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.
To save the PIN code you can for example scramble it and store it in Block 0 of the EEProm.
Modifying the PIN number to 0, will deactivate the protection with PIN numbers.

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 (see Table 8: Electrical specifications)	32	50	N / 15.1 Amp	3.3 A	3.3 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 5: 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 Flow 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

7.3. 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 01 03 00 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 6: Hopper Status .

Hopper Status		
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	Not used	0
B7	1 = Payout disabled	1

Table 6: Hopper Status

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 the hopper 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. These flags 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 the dispensing commands are transmitted within 333ms after receiving the ACK response from the Reset command, the opto flags will be cleared and the hopper may be started again to start a new payout.

In our example the status byte (80) tells us that the hopper is dissabled. If no opto error flags are set, the ENABLE HOPPER 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. Note that a coin may be stuck in the coin exit port if the power fails just before a coin enters the coin exit port. Since the power is gone, the hopper will not be able to eject this coin. After power up, transmit a dispense command within 333ms after a reset command to eject this coin, before the opto sensor generates an opto blocked error again.

⇒ 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 when the power supply is 12V and below 18V during 20ms when the power supply is 24V.

The time to save all coin counters in EEProm memory may take up to 90ms. 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.

Transmitting the Emergency Stop command during normal operation (e.g. no power down), the hopper will stop as soon as possible in a controlled way. The hopper will stop when the next coin (not the coin that is being paid) is ejected fully. After transmitting the Emergency Stop command, always retreive the nrCoinsPaid and nrUnPaid counters with the Request Hopper Status command to get the actual number of coins paid since the hopper motor stopped.

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 is set to encrypted (dipswitch 8 ON, MC encrypted), 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 dipswitch 8 is set to OFF (ITA serial), 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.

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 dipswitch 8 is set to OFF (ITA serial), 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. When the event counter is 255, the next payment command sets the counter to 1. After a reset, the 4 bytes is reset to 0

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 levelstatus.
The bits of levelstatus 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 7: LevelStatus bit definition

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

The High Level sensor is fitted bit will however always be low, and cannot be used to check if the high level sensor is present.

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 is erased with a reset. Power off will not reset te counter

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.

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. (dipswiich 8 OFF)
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 is MC encrypted, dipswitch 8 ON, 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 is ITA serial, dipswitch 8 OFF", 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 ITA
                                                    // serial
    if (HopperType == "MC encrypted" )                     // use security bytes?
    {
        DispenseCoins(HopperAddress, SecurityBytes, NrCoinsToPay);
    }
    else if (HopperType == "ITA serial")     // use dummy bytes? (security disabled)
    {
```

```
        DispenseCoins(HopperAddress, DummyBytes, NrCoinsToPay);
    }
    else if (HopperType == "ITA serial")     // use serial nr as dispense key?
    {
        DispenseCoins(HopperAddress, SerialNr, NrCoinsToPay); // serial number already
requested during initialization
    }
}
else    // hopper errors
{
    ResetHopper();            // send reset command (set receive timeout to 100 ms)
    // in case of opto errors, send dispense commands within 333 ms after receiving ACK
    // from 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
```

7.4. 3.4 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:
- 42PP0520 (dual supply: 5VDC and 12VDC)
- 42PP0530 (dual supply: 5VDC and 24VDC)

When the voltage drops below the POWER_FAIL_TRESHOLD (see Table 8: Electrical specifications) 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 8.0V 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 3.3A during 160ms, the motor will start reversing in order to un-jam the hopper. During anti-jamming, the current my rise to levels of 6 Amp peak.

7.5. 3.5 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).

8. Electrical, Timing and Environmental Specifications

Parameter	Min	Typ @ 12V	Typ @ 24V	Max	Units
Supply voltage	11	12	24	26	Vdc
Idle current		30	30		mA
Motor_start current		2.2	2.2	5	A
Motor_running_current		0.5	0.6	0.9	A
Vtrip (Power fail threshold)		8.0 during 20ms	8.0 during 20ms		Vdc
Motor_current_reverse level		> 3.3 A during 160 ms	> 3.3 A during 160 ms		A
Thermal fuse protection		5.5	5.5		A

Table 8: Electrical specifications

Parameter	Description	Value	Units
Brate	Serial communication speed	9600	baud
Trxout	Receive data timeout	25	ms
TPinit	Power up initialisation time	630	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
TResetAck	ACK delay after receiving a reset command	max 90	ms
TResponse	Command response time (except for command headers 1, 253, 252)	max 2	ms

Table 9: Timing specifications

Print this on A5 paper for correct size.

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 10: EEProm Memory Description

Storage temperature	-20º C to +70º C
Operating temperature	0º C to +50ºC
Storage humidity	10% to 95% non-condensing
Operating humidity	10% to 75% RH
General	Keep coin exit clear from any obstacles

Table 11: Environmental specifications

9. Warranty

We thank you for the purchase of this Suzo product. If you require warranty for this product you can contact Suzo International. Before contacting Suzo we advise to carefully read the product manual.

9.1. Your warranty

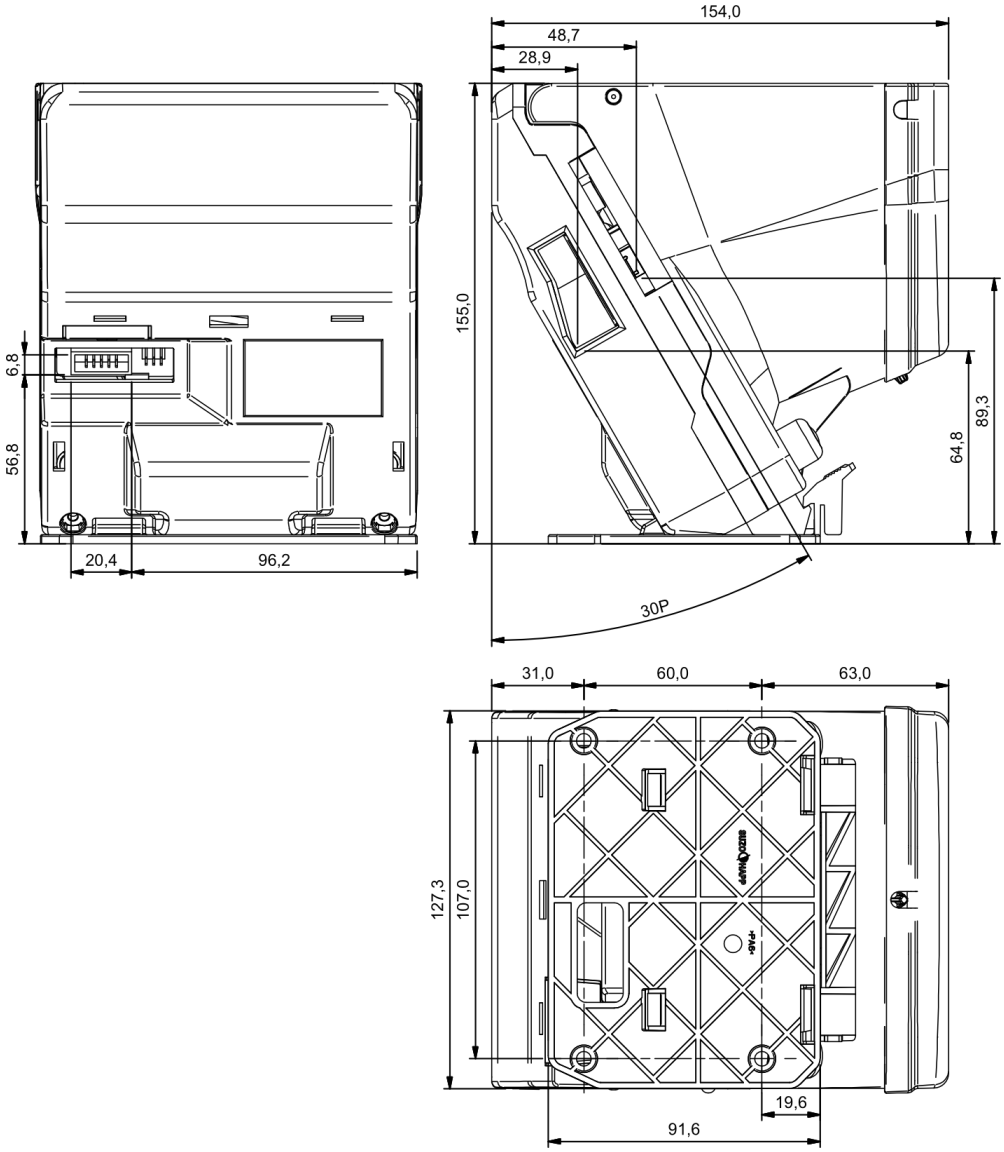
Suzo guaranties that the product for the period of ONE YEAR from the original purchase date will be free of material or fabrication errors. When material or fabrication errors do occur (as judged by Suzo) within the warranty period, Suzo will replace or repair the defective parts without labour or parts cost. The following restrictions apply. All replaced products or parts become property of Suzo International.

9.2. Conditions

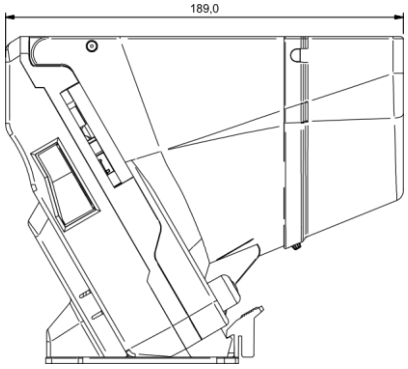
1. Warranty can only apply when the original invoice (with model number and purchase date) together with the defective product is presented within the warranty period.
2. Suzo is under no obligation to perform warranty when invoice is not present or unreadable.
3. Warranty does not apply when model number of serial of product are changed, removed or made unreadable.
4. Warranty does not cover the risk to the product during transportation to and from Suzo International
5. Warranty does not cover:
    a. Regular maintenance or replacement of parts due to normal wear.
    b. Damage of defects caused by operation or treatment of the product not considered as normal use.
    c. Damage or adaptations to the product caused by faulty operation or use that is not within the machines original capabilities.
6. Warranty is only performed at the Suzo Technical office unless other official agreements apply for instance as agreed upon in a maintenance contract.

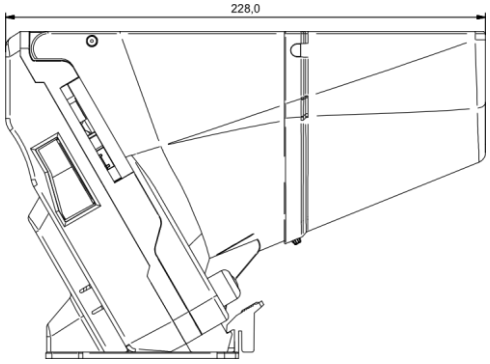
10. Dimensions

10.1. General dimensions



10.2. Medium and large Extensions





11. Revision History

Revision	Date	Comment	By
0.1	09 Jul. 2010	Pre release	E.S.
0.4	29 Sept 2010	Drawings and exploded views	M.M.
0.5	29 Nov 2010	Update manual	M.M.
0.6	16 Feb 2011	Update commands	E.S.

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.

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.

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 powersupply.