

ECM

User Manual





Copyright © 2012 - 2021 SmarAct GmbH

Specifications are subject to change without notice. All rights reserved. Reproduction of images, tables or diagrams prohibited.

The information given in this document was carefully checked by our team and is constantly updated. Nevertheless, it is not possible to fully exclude the presence of errors. In order to always get the latest information, please contact our technical sales team.

SmarAct GmbH, Schuette-Lanz-Strasse 9, D-26135 Oldenburg
Phone: +49 (0) 441 - 800879-0, Telefax: +49 (0) 441 - 800879-21
Internet: www.smaract.com, E-Mail: info@smaract.com

Document Version 2.9, Build 55

TABLE OF CONTENTS

1 Introduction.....	5
1.1 Units.....	5
1.2 Activating And Deactivating Units.....	6
1.2.1 Automatic Unit Deactivation In Case Of Errors.....	7
1.3 Adding and Configuring Units.....	7
1.4 Device Locators.....	8
1.4.1 USB Device Locator Syntax.....	8
1.4.2 Network Device Locator Syntax.....	8
1.4.3 Device Locators of Connected USB Controllers.....	9
2 The LCD Display Module.....	10
2.1 Menus Overview.....	10
2.2 Selecting Menus and Fields.....	11
2.3 Editing of Settings.....	11
2.4 The Units Menu.....	13
2.5 Configuring the Network Interface.....	13
2.6 Configuring the Serial Port.....	14
2.7 Restoring Default Settings.....	14
2.8 Saving Diagnostics Data.....	14
2.9 Updating the ECM Firmware.....	15
2.10 Updating MCS Controller Firmware.....	15
3 Communicating With the ECM.....	17
3.1 Connecting.....	17
3.1.1 Network.....	17
3.1.2 Finding the Network Interface MAC Address.....	17
3.1.3 Serial Port.....	17
3.2 Command Syntax.....	17
3.3 Command Replies.....	17
3.4 Number Formats.....	18
3.5 Getting Online-Help From the System.....	19
4 System Commands.....	20
%help – Show System Commands Overview.....	20
%code? – Show Status Code Description.....	21
%info – Show Information.....	22
%activate-unit – Activate Unit.....	24
%deactivate-unit – Deactivate Unit.....	25
%unit-activated? – Get Unit Activation Status.....	26
%add-unit – Add Units.....	27
%remove-unit – Remove Units.....	28
%config-unit – Configure Unit.....	29
%unit?, %unit – Get Selected Unit/Select Unit.....	30
%set – Set System Properties.....	31
%get – Read System Properties.....	32
%echo – Reply Text to Sender.....	33

%save-diagnostics – Save Diagnostics Data.....	34
5 SmarPod Units.....	35
5.1 SmarPod Commands.....	35
help – Show SmarPod Commands Overview.....	35
ref?, ref – Get Referenced-Status/Start Referencing.....	36
confsys – Configure System for SmarPod Model.....	37
models? – List Supported SmarPod Models.....	38
cal – Calibrate Sensors.....	39
rea? – Is Pose Reachable?.....	40
mov – Move.....	41
stop – Stop Movement.....	42
stoh – Stop Movement And Hold Position.....	43
stby – Activate Standby Mode.....	44
mst? – Get Move Status.....	45
pos? – Get Physical Pose.....	46
piv?, piv – Get/Set Pivot-Point.....	47
pvm?, pvm – Get/Set Pivot-Mode.....	48
csy?, csy – Get/Set Coordinate System.....	49
spz – Set Current Pose as Zero.....	50
sen?, sen – Get/Set Sensor-Mode.....	51
frq?, frq – Get/Set Maximum Frequency.....	52
vel?, vel – Get/Set Speed.....	53
acc?, acc – Get/Set Acceleration.....	54
get, set – Get/Set a SmarPod Property.....	55
5.2 SmarPod Unit Properties.....	56
5.3 SmarPod Unit Error Codes.....	58
6 MCS Units.....	60
6.1 MCS Commands.....	60
help – Show MCS Commands Overview.....	60
nch? – Get Number Of Channels.....	61
sta? – Get Channel Status.....	62
sty – Set Sensor Type.....	63
sty? – Get Sensor Type.....	64
cal – Calibrate Sensor.....	65
ref?, ref – Get Referenced Status/Start Referencing.....	66
htm – Set Hold Time.....	67
frq – Set Frequency.....	68
vel?, vel – Get/Set Closed-Loop Speed.....	69
sfd – Set Safe Direction.....	70
sen?, sen – Get/Set Sensor Mode.....	71
mpa, mpr – Move To Absolute/Relative Position.....	72
maa, mar – Move To Absolute/Relative Angle.....	73
mst – Step-Move Positioner.....	74
stop – Stop One Or All Channels.....	75
pos? – Get Position.....	76
ang? – Get Angle.....	77
6.2 MCS Unit Command Error Codes.....	78

6.3 MCS Unit Channel Status Codes.....	81
7 Appendix.....	82
7.1 Example Command Sequence.....	82
7.2 System Error Codes.....	83

1 INTRODUCTION

An SmarAct *Embedded Control Module* (ECM) works as a command interpreter for SmarAct positioning systems like SmarPods and actuators connected to MCS controllers. Clients can connect to an ECM via ethernet or RS232 serial cable. Its is available as rack module, compatible with SmarAct 19" racks, or as table-top device with an integrated LCD display.

For a detailed description of installing and connecting an ECM, please refer to the *ECM Hardware-Manual*.

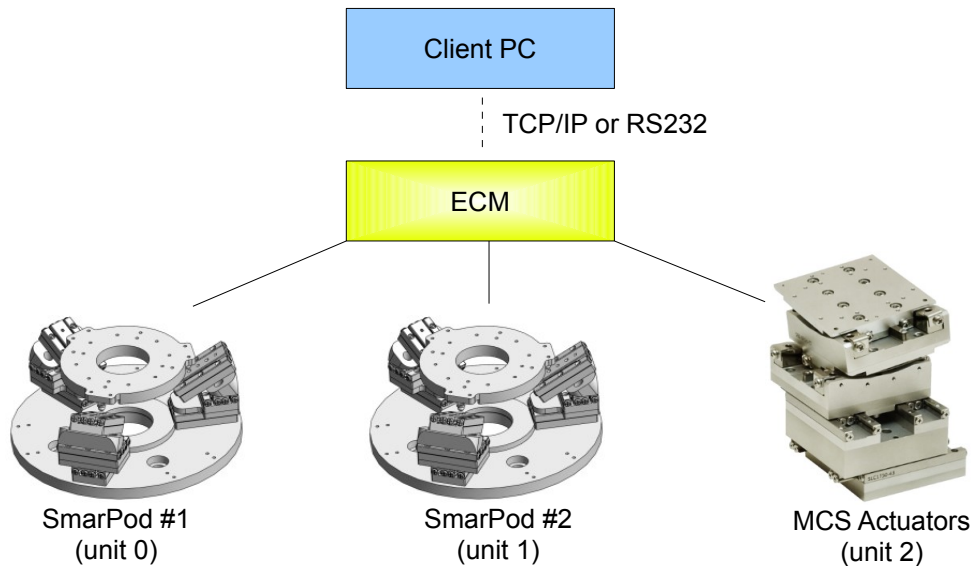
In table-top or multi-rack configurations, the external devices should be switched on before the ECM or they will not be detected by the ECM automatically on startup. The activation status can be checked with the system command `%info units` or in the *Units* menu of the LCD display. To activate units whose controllers are switched on later, see 1.2 "Activating And Deactivating Units".

In this document the following notation is used: In code examples arrows indicate message directions:

← ref?	an arrow pointing to the left means: outgoing command from the client PC to the ECM
→ 0	an arrow pointing to the right means: an answer received from the ECM

1.1 Units

A *unit* is an addressable device connected to the ECM. One ECM can control several units. Different unit types provide different command sets. The ECM in the following image controls two SmarPods, configured as unit 0 and unit 1, and one MCS controller with several actuators, configured as unit 2:



Before sending commands to a unit it must be selected as the command target. Use the system command `%unit` to select a unit. All following commands that are not system commands are routed to that unit.

Example:

← <code>%unit 0</code>	select unit 0 (activates the SmarPod command set)
→ <code>!0</code>	
← <code>mov 0 0 0 0 0 -4</code>	move SmarPod #1 to -4° around the z axis
→ <code>!0</code>	
← <code>%unit 2</code>	select unit 2 (activates the MCS command set)
→ <code>!0</code>	
← <code>mpr 0 -2m</code>	move actuator on MCS channel 0 by a distance of -2 millimeters
→ <code>!0</code>	

1.2 Activating And Deactivating Units

Only activated units can process commands. When sending a command to a deactivated unit, an error is returned.

When the ECM controller is booting, it tries to activate all configured units automatically. It is possible that the activation of a unit fails, for example, if no connection to the unit hardware can be established. This can happen if an external MCS is powered on after the ECM. It can also indicate a hardware defect or a wrong unit configuration.

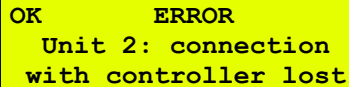
Units can be activated and deactivated in the *Units* menu of the LCD display (see 2.4 "The Units Menu") or with system commands.

An activated unit can be deactivated with the `%deactivate-unit` system command. The unit activation status can be checked with the `%info units` command which list all units, or the `%unit-activated?` command, which returns the activation status for one unit. Deactivated units can be activated with `%activate-unit`.

A unit should be deactivated before physically disconnecting the unit hardware from the ECM.

1.2.1 Automatic Unit Deactivation In Case Of Errors

An activated unit checks the connection to its MCS controller periodically. If a communication error is detected, the unit status is set to *error* and subsequent unit commands that require a working connection will fail. At the same time an error message is shown on the display:



```

OK      ERROR
Unit 2: connection
with controller lost
  
```

The dialog can be closed with *OK*. To continue working with the unit, it must be activated again.

1.3 Adding and Configuring Units

New units are created with system command `%add-unit`. The command expects the type of the unit in the first argument. In the second, optional argument, a unit index can be passed. If a unit index is specified, the new unit will be created at that index if the index is not already used. Otherwise an error code is returned. The index must not be greater than the last used index + 1 and less or equal the maximum unit index (127). If no unit index is specified, the system automatically assigns the next free index to the unit. If the unit is successfully created, the index of the unit is returned.

A newly added unit is neither configured nor activated. This has to be done with the `%config-unit` and `%activate-unit` commands before the unit can be used.

Let's assume that an ECM has some configured units:

```

← %info units
→ u0: type=mcs mcs=usb:id:3784864961
    (error: activation failed: no systems found)
    u1: type=smarpod model=10001 mcs=usb:id:7611829102 (active)
    u3: type=smarpod model=10001 mcs=usb:id:1139267180 (active)
  
```

Units 0, 1, and 3 exist, but unit index 2 is unused. To add a new unit of type "smarpod" at the next free index (2), we write the command

```

← %add-unit smarpod
→ 2                the new unit index
  
```

The unit list is now:

```

← %info units
→ u0: type=mcs mcs=usb:id:3784864961
    (error: activation failed: no systems found)
    u1: type=smarpod model=10001 mcs=usb:id:7611829102 (active)
    u2: type=smarpod model=0 mcs=unspecified (deactivated)
    u3: type=smarpod model=10001 mcs=usb:id:1139267180 (active)
  
```

Now we configure and activate the new unit. Please refer to section 1.4 for a discussion of device locators. The SmarPod MCS controller is connected via network to the ECM, so we set the option "mcs" to the network system locator:

```

← %config-unit 2 model 10007
→ !0
← %config-unit 2 mcs network:192.168.47.101:2000
→ !0
  
```

```
← %activate-unit 2
→ !0
```

The new unit could also have been added with specified index 4 with command `%add-unit smarpod 4`, which is the next free index after the highest index in use.

1.4 Device Locators

The ECM needs to know where to find a unit's controller hardware. The address of a controller is specified as a *device locator* which contains the controller's interface type and other identification data.

1.4.1 USB Device Locator Syntax

USB Device Locators

The general USB locator format is:

```
usb:<address>
```

The options for the `<address>` part are slightly different for MCS and MCS2 controllers:

MCS:	Examples:
<code>usb:id:<id></code>	<code>usb:id:3761627481</code>

MCS2:	<code>usb:sn:<serialnumber></code>	<code>usb:sn:MCS2-00001234</code>
-------	--	-----------------------------------

`<id>` is the first part of a MCS (1) controller serial number which is printed on the MCS controller. For example, a MCS type and serial number label should show something similar to "S/N : 12345678.0987" where the 12345678 part is the ID. This option is not available for MCS2 controllers.

`<serialnumber>` is the serial number of an MCS2 controller. This option is not available for MCS controllers.

1.4.2 Network Device Locator Syntax

The general network locator format is:

```
network:<address>
```

The options for the `<address>` part are slightly different for MCS and MCS2 controllers:

MCS:	Examples:
<code>network:<ipv4>:<port></code>	<code>network:192.168.1.200:5000</code>

MCS2:	<code>network:<ipv4></code>	<code>network:192.168.1.200</code>
	<code>network:sn:<serialnumber></code>	<code>network:sn:MCS2-00001234</code>

`<ipv4>` is an IPv4 address which consists of four integer numbers between 0 and 255 separated by a dot. `<port>` is an integer number. The locator `network:192.168.1.200:5000` describes an MCS controller with the IPv4 address 192.168.1.200 and TCP port 5000. The locator `network:sn:MCS2-00001234` can be used for the MCS2 with the serial number MCS2-00001234.

For MCS controllers the TCP port must be included and is usually 5000 (if the network address of the controller has not been reconfigured). For MCS2 controllers do **not** add a port to the IPv4 address.

Do not add leading zeros to the numbers in the IPv4 address as these would be interpreted as octal numbers. For example, the IPv4 addresses 192.168.030.200 and 192.168.30.200 are **not** the same!

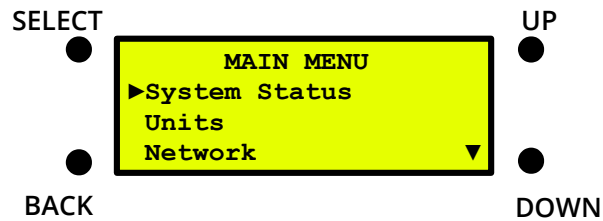
For example, the locator network:192.168.1.200:5000 addresses a device with the IPv4 address 192.168.1.200 and TCP port 5000.

1.4.3 Device Locators of Connected USB Controllers

Command `%info mcs` returns an overview of all MCS controllers which are currently connected to the ECM via USB interface and not assigned to an activated unit. Controllers of activated units and controllers with a network interface are not listed. The list contains the device locator of each controller which can be used when adding or reconfiguring a unit (see 1.3).

2 THE LCD DISPLAY MODULE

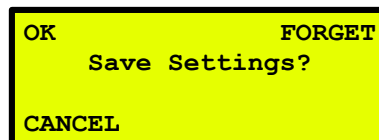
This section describes the 20x4 characters LCD display with 4 buttons.



The four buttons at the corners of the LCD display have the following default functions:

Default Button Functions	
<i>SELECT</i>	open sub-menu / execute a function
<i>BACK</i>	go back one menu level / cancel a function
<i>UP</i>	move up / increment value
<i>DOWN</i>	move down / decrement value

The function of a button changes if a function name is shown in the corner next to that button:



2.1 Menus Overview

The following menus and system functions are available:

MAIN MENU top level menu

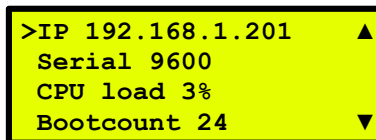
<i>System Status</i>	shows system status details
<i>Units</i>	opens a list of configured units (see 2.4)
<i>Unit #</i>	editor and status display for one unit
<i>Network</i>	for configuring network I/O (see 2.5)
<i>Serial Port</i>	for configuring the serial I/O (see 2.6)
<i>Utilities</i>	sub-menu with system utilities
<i>Update Firmware</i>	install an ECM firmware update (see 2.9)
<i>Restore Defaults</i>	reset the ECM to factory defaults (see 2.7)
<i>Save Diagnostics</i>	save system diagnostics to a USB stick (see 2.8)
<i>MCS Controllers</i>	(see 2.10)
<i>Update MCS</i>	update the firmware of all MCS that have an old firmware
<i>Update MCS (force)</i>	update the firmware of all MCS regardless of fw-version
<i>Device Info</i>	shows the ECM firmware version and serial number

2.2 Selecting Menus and Fields

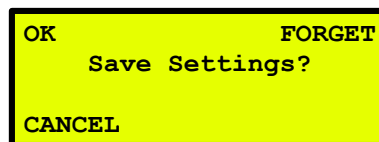
Menus are lists of text or function fields. Some menus are longer than the height of the display. In this case the display shows filled up- and down-arrows at the upper and lower right corners. If arrows are visible, the menu can be scrolled with *UP* and *DOWN* buttons.

The navigation cursor marks the selected field. A filled triangle ► indicates that the field can be either edited or will execute a function or opens another menu on clicking *SELECT*.

An open triangle > is displayed if the selected field contains only non-editable text and it does not execute a function when pushing *SELECT*:

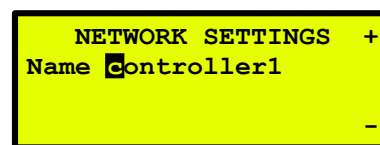
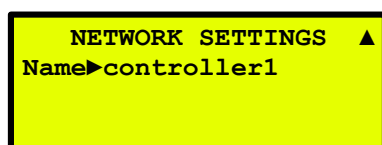


If the field is a sub-menu function, the sub-menu is opened on pushing *SELECT*. If it is an editable field, the menu changes from navigation mode to edit mode. See section 2.3 "Editing of Settings". To close a sub-menu click on the *BACK* button. If a sub-menu contains settings fields that have been edited, a dialog screen asks to confirm the changes. The changed settings can be accepted and activated (*OK*) or discarded (*FORGET*). If *CANCEL* is pushed, closing of the menu is canceled.



2.3 Editing of Settings

If the *SELECT* button is pushed on an editable field, the menu changes from *navigation mode* to *edit mode*. In edit mode + and - symbols are shown in the right corners and the navigation cursor is no longer visible. Instead, one of the characters of the settings value is blinking (*edit cursor*):



If the field consists of more than one sub-field (e.g. an IPv4 address field), pushing the *SELECT* button again selects the next sub-field. There are different editable field types:

- **a list of presets:** + and - cycle through a list of presets.

- **integer numbers:** + and - increment and decrement the value. Hold the UP or DOWN button to increase the speed of the value in- or decrements.
- **text fields:** when editing a text field, pushing *SELECT* moves the edit cursor one place to the right. + and - cycle through the characters under the cursor. When the *SELECT* button is hold, the + and - symbols are replaced by > and < until *SELECT* is released. In this mode the *UP* button (>) inserts a new character and the *DOWN* button (<) deletes the character under the cursor.

To stop editing a field and return to navigation mode push the *BACK* button.

2.4 The Units Menu

In the *Units* menu all units are listed. A line in the list show the index, the current activation status and the type of one unit.

```

      UNITS
u0! smarpod
►u1◊ smarpod
u2◆ mcs
  
```

Next to the unit index, a symbol shows the current activation status of that unit (see 1.2 "Activating And Deactivating Units"):

Activation Status Symbols

!	the last activation has failed
◊	the unit is deactivated
◆	the unit is activated

To open an editor for a unit move the navigation cursor up or down until the unit is selected and push the *SELECT* button.

```

      UNIT 0 ▲
►Type: mcs
  Status: activated
  Deactivate
  
```

In the editor the unit type and activation status are displayed. Push the *SELECT* button on the *Status* line to show the reason why the activation has failed:

```

activation failed:
no systems found
  
```

With the functions *Activate* and *Deactivate* the unit can be activated or deactivated.

It is not possible to add new units or remove existing units from this menu. Also, the configuration of a unit cannot be changed here. This must be done with the respective system and units commands.

2.5 Configuring the Network Interface

In the *Network* menu the network interface can be configured. In field *Mode* one of the following operation modes can be set:

Network Configuration

<i>disabled</i>	the network interface is disabled.
-----------------	------------------------------------

<i>DHCP</i>	the system will contact your DHCP server to get an IPv4 address.
<i>fixed address</i>	the IPv4 address can be entered in the sub-menu IP Address.

A network name for the system can be entered in the field *Name*.

The TCP port is always 2000. The new settings are activated when the configuration page is closed and confirmed with *OK*.

See also 3.1 "Connecting".

2.6 Configuring the Serial Port

The serial port can be configured in the sub-menu *Serial Port*. The serial port can be enabled (default) or disabled. In the field *Baudrate* one of the presets can be selected.

The new settings are activated when the configuration page is closed and confirmed with *OK*.

2.7 Restoring Default Settings

The controller default configuration can be restored with the function *Restore Defaults* from the *Utilities* menu. This resets the network and serial port configurations as well as the system properties like *number-format* and deletes all units. The device will reboot automatically.

The function does not delete basic system data like the boot count and the system log.

After the settings are restored, the device may not be accessible from the network or over the serial port anymore. In this case, the interfaces must be configured again.

2.8 Saving Diagnostics Data

In cases of problems with the system hard- or software or when a system update has failed, diagnostics information can be saved to a USB memory stick and emailed to SmarAct for inspection. Plug a USB memory stick into one of the system USB ports and select the function *Save Diagnostics* in the *Utilities* menu.

A file named similar to *ECM.00000001_BC10.diag* or *ECM.00000001_BC10_2.diag* will be saved, where *00000001* is the serial number and *BC10* the boot count of the ECM. If more than one diagnostics file has been created without rebooting the system, the *_2* part in the file name indicates the number of the diagnostics file. When you are asked to send a diagnostics file to SmarAct, please always send the latest file.

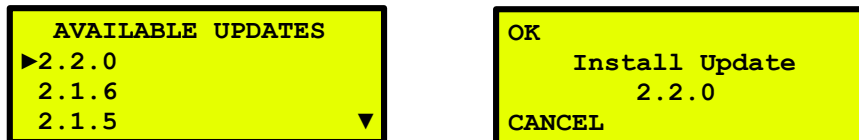
This function can also be executed by sending the `%save-diagnostics` command.

2.9 Updating the ECM Firmware

The ECM firmware update requires a FAT32 formatted USB memory stick.

ECM firmware updates can be installed with the *Update Firmware* function in the *Utilities* sub-menu. To update the firmware, copy the firmware file (named *ecm-2.0.0_2016-02-08_145315.fw* or similar) to the USB memory stick and plug it into one of the ECM USB ports.

When *Update Firmware* is selected, firmware updates found on the USB stick are listed. From that list, select a version with *UP/DOWN* and click on *SELECT* to start the update or *BACK* to return to the *Utilities* menu. On the next screen confirm or cancel the update:



Downgrading to older firmware versions is not supported.

During the update, the display shows the text "UPDATING...". At the end of the update the ECM restarts automatically.

If the update fails, an error message is displayed and the controller keeps the last firmware version. Information about the failure may be contained in the system log, which can be obtained by %info Log. Reason for failed updates can be damaged firmware files or a mismatch between the new and the installed firmware version. In other cases you should inform SmarAct about this.

An ECM firmware update does not update the firmware of connected MCS controllers. To update the MCS controller firmware, see 2.10 "Updating MCS Controller Firmware".

2.10 Updating MCS Controller Firmware

Only MCS 1 firmware can be updated from the ECM. For updating MCS2 controller firmware please follow the instructions in the MCS2 documentation.

MCS controllers connected to the ECM have their own firmware which is independent from the ECM firmware (to update the ECM firmware see 2.9 "Updating the ECM Firmware").

To update the MCS controllers you need put a MCS firmware folder in the top-level directory of a USB memory stick. If the firmware is compressed in a ZIP archive, the files must be uncompressed. Ensure that the uncompressed folder contains the following files:

folder (copy this folder to the USB stick)
update.xml
 ...other files and sub-folders....

Plug it in one of the USB ports of the ECM.

Update functions can be found in the *MCS Controllers* menu under the *Utilities* menu. The ECM updates the MCS firmware in two different ways:

- *Update MCS* starts a “smart” update that updates only components which are older than the firmware on the USB memory stick. This is the recommended method for updating because it is the fastest and it guarantees that the new firmware is never older than the firmware currently installed.
- *Update MCS (Force)* updates all components of all MCS controllers regardless of their versions. This can take longer than the function above. To downgrade the MCS firmware to an older version this function must be used.

Note, that the system **must not be turned off until the update has finished**. Otherwise the firmware update can be incomplete and the controllers may not work as expected or will not be recognized by the ECM. (In such a case they would have to be send back to SmarAct for repair). During the update, buttons on the controllers should not be pushed, cables and USB devices not be plugged in or removed. Do not send commands to the controller during the update.

When the update has finished, the MCS controllers must be restarted to activate the new firmware. If ECM and MCS controllers are integrated in a 19” rack, the rack must be restarted. If they are table-top units, they must be restarted separately.

3 COMMUNICATING WITH THE ECM

3.1 Connecting

3.1.1 Network

You can connect to an ECM with any program that can send and receive text over TCP, like a terminal program or commands like *telnet* or *netcat*. The program must connect to the system's TCP port 2000:

```
telnet 172.16.16.110 2000
```

See 2.5 “Configuring the Network Interface” for information about the configuration of the network interface.

3.1.2 Finding the Network Interface MAC Address

The MAC address of the network interface can displayed in the *Network* menu under *Show MAC Address* or by executing the system command `%info network`.

3.1.3 Serial Port

The connecting program must set the same serial port protocol configuration as the controller. The controller's RS232 settings are: 8N1 (8 databits, no parity, one stop-bit). The baudrate is configurable (see 2.6 - “Configuring the Serial Port”).

After the connection is established via network or serial port there is no automatic response from the controller. You should check the connection, for example by calling one of the system commands `%info`, `%help` or `%echo`.

3.2 Command Syntax

Commands have the following syntax:

```
<command-name>[<space><parameter1>[<space><parameter2>...]]<linefeed>
```

The number of parameters depends on the command. Each command must be followed by one linefeed character (0x0a).

Every command returns an answer to the caller. An answer is followed by a line-end. If the answer has more than one line, they are separated by line-ends. The format can be set with `%set lineend-format`.

3.3 Command Replies

Some commands return a status code which indicates whether the command was successful or, if not, which error has occurred. Returned status codes are prefixed with an exclamation-mark:

```
← %unit 0
→ !0                no error
```

```
← %zzzquax
→ !10003 "unknown command"
```

Query commands return text if they succeed and an error code if not:

```
← pos?
→ 0 0 650u 0 0 0 SmarPod pose data
```

or:

```
← pos?
→ !550 "SmarPod not referenced"
```

Status codes are either system status codes (≥ 10000) or unit status codes (< 10000). To get information about a status code you can use the `%code?` command. For a list of system error codes see 7.2. The error codes for the units can be found in the chapters for the respective unit types.

3.4 Number Formats

The system can understand and generate numbers in different formats:

- **scientific format:** e.g. 4.4532E-6 or 4.4532e-6
- **fixed format:** e.g. -15.453234
- **SI format:** e.g. 450n, 5.5k, 2m, -75u

In SI format a number can be followed by one of these symbols:

SI Format Symbols		
Symbol	SI-Prefix	Factor
<i>P</i>	peta...	10^{15}
<i>T</i>	tera...	10^{12}
<i>G</i>	giga...	10^9
<i>M</i>	mega...	10^6
<i>k</i>	kilo...	10^3
<i>m</i>	milli...	10^{-3}
<i>u</i>	micro...	10^{-6}
<i>n</i>	nano...	10^{-9}
<i>p</i>	pico...	10^{-12}

which modify the value by the listed factors. Thus '2u' equals 0.000002 or 2E-6. (Remember, that the suffix 'm' in numbers like '2m' means *milli-*, not *meters!*).

Between the number and the following SI symbol must be no space.

When passing numbers as command parameters to the system any format can be used. The format of numbers returned by the system can be configured with the system command `%set number-format`.

3.5 Getting Online-Help From the System

Once connected with the controller, help texts for the system and unit commands can be shown. To get an overview of the system commands enter `%help`.

To get the command overview for the currently selected unit enter `help`.

4 SYSTEM COMMANDS

System commands have an effect on the system as a whole. They are prefixed with the % symbol.

%help – Show System Commands Overview

Syntax:

`%help`

Description:

This function returns a list of all system commands with a short description.

%code? – Show Status Code Description

Syntax:

%code? c

Description:

This function returns a description of the error/status code n. Codes < 10000 are unit specific codes. Their meanings depends on the type of the unit. Codes ≥ 10000 are system codes. An exception is status code 0, which means OK for units and system commands.

For a list of status codes that can be returned see 7.2 “System Error Codes”.

Example:

```
← %code? 10003  
→ unknown command
```

%info – Show Information

Syntax:

%info selector

Description:

This function returns information about the system. selector must be one of the following:

- status: returns information about the ECM system status.
- units: returns a list of the units and their status.
- unit-types: returns the unit types supported by this version of the ECM.
- unit-options: Returns the configuration options by type (see also %config-unit)
- network: returns detailed information about the network interface status and configuration.
- log: returns the system log which contains events like successful or failed firmware updates.
- device: returns the information about the device, including the serial number, the product code and the current firmware version.
- mcs: returns information about the connected MCS (Modular Control Systems) controllers.

Examples:

```
← %info device
→ Device serial number: ECM.00001772
   Device product code: ECM-2.0-Rack
   Firmware version: 2.0.1
```

```
← %info status
→ Status: ready
   Bootcount: 31
   Uptime: 3h:1715s
   CPU Temperature: 26.80 degrees Celsius
   CPU Load: 2.00%
   Memory Available: 91.2%
   Network IO: ok (277 bytes read, 11006 bytes written)
   IP address: 192.168.42.7
   Connected network client: 192.168.42.112
   Serial IO: ok (0 bytes read, 0 bytes written)
   Display: ok
```

```
← %info log
→ System Log for ECM.00001772
   INFO: Firmware update version 2.0.2
         Boot count 10, uptime 11m26s
         The firmware update was completed successfully.

   ERROR: Firmware update version 2.1.2
          Boot count 11, uptime 17m27s
          The installed firmware must be at least version 2.1.0, is 2.0.2.

   INFO: MCS usb:id:3416818407 firmware update
          Boot count 151, uptime 9m12s
          Firmware of MCS usb:id:3416818407 was successfully updated.
```

```
← %info units
→ Units:
  u0: type=smarpod model=10003 mcs=usb:id:3416818407
      (error: activation failed: no controller systems found)
  u1: type=mcs mcs=usb:id:1285918560 (deactivated)
```

%activate-unit – Activate Unit

Syntax:

%activate-unit n

Description:

This command activates unit n. If the unit cannot be activated, an error is returned.

To send commands to an activated unit, it must be selected first with the %unit command.

See also 1.1 “Units”.

Example:

```
← %activate-unit 0
→ !0
← %unit-activated? 0
→ 1
```


%deactivate-unit – Deactivate Unit

Syntax:

%deactivate-unit n

Description:

This command deactivates unit n. After deactivation the unit will not execute commands.

See also 1.1 “Units”.

Example:

```
← %deactivate-unit 0
→ !0
← %unit-activated? 0
→ 0
```

%unit-activated? – Get Unit Activation Status

Syntax:

%unit-activated? n

Description:

This command returns the current activation status of unit n. The command returns 1 if the unit is activated and 0 if not.

See also 1.1 “Units”.

Example:

```
← %deactivate-unit 0  
→ !0  
← %unit-activated? 0  
→ 0
```

%add-unit – Add Units

Syntax:

```
%add-unit type [unit]
```

Description:

This command creates a new, unconfigured unit. The type must either be „mcs“ or „smarpod“. If the optional unit index is not specified, the unit's index will be the first free index available. Otherwise, the specified index, which must be free, is used. The specified unit index must not exceed one above the highest available unit index and must be less or equal the maximum unit index (127). The created unit is unconfigured and deactivated. If the command was successful, the new unit index is returned.

For a more detailed discussion about adding units see 1.3 "Adding and Configuring Units"

See also 1.1 "Units" and %config-unit – Configure Unit.

Example:

```
← %info-units
→ units:
  u0: type=mcs mcs=usb:id:3784864961 (deactivated)
  u3: type=mcs mcs=usb:id:3784864960 (active)
← %add-unit mcs
→ 1
← %add-unit mcs 3
→ !10100
← %add-unit smarpod 4
→ 4
← %info-units
→ units:
  u0: type=mcs mcs=usb:id:3784864961 (deactivated)
  u1: type=mcs mcs=unspecified (deactivated)
  u3: type=mcs mcs=usb:id:3784864960 (active)
  u4: type=smarpod mcs=unspecified model=0 (deactivated)
```

%remove-unit – Remove Units

Syntax:

```
%remove-unit unit
```

Description:

This command removes a unit. The unit index must belong to an existing deactivated unit.

See also 1.1 “Units”.

Example:

```
← %info-units
→ units:
   u0: type=mcs mcs=usb:id:3784864961 (deactivated)
   u3: type=mcs mcs=usb:id:3784864960 (active)

← %remove-unit 0
→ !0

← %info-units
→ units:
   u3: type=mcs mcs=usb:id:3784864960 (active)
```

%config-unit – Configure Unit

Syntax:

```
%config-unit unit property value
```

Description:

With this command properties of a unit can be configured. For example, it can be necessary to change the addresses of MCS controllers, if MCS controllers (or parts of them) are replaced.

The unit to configure must be deactivated. To deactivate an active unit, use the %deactivate-unit command.

See also 1.1 “Units”.

Parameters:

The parameter unit selects the unit. The parameters property and value depend on the type of the unit. Valid parameters by unit type may also be obtained with the %info unit-options command.

Units Configuration Properties	
Property	Values
SmarPod Units	
<i>model</i>	A SmarPod model code. E.g. 10001 (SmarPod 110.45 S). When the model is been changed, the SmarPod command confsys must be executed as soon as the unit is activated in order to configure the MCS controller for new SmarPod model.
<i>mcs</i>	The system locator of the MCS controller that controls the SmarPod.
MCS Units	
<i>mcs</i>	The system locator of the MCS controller.

Example:

```
← %deactivate-unit 0
→ !0

← %info unit-options
→ Type dependent unit configuration options:
   mcs      : 'mcs'
  smarpod: 'mcs, model'

← %config-unit 0 mcs usb:id:1234567890
→ !0

← %activate-unit 0
→ !0
```

%unit?, %unit – Get Selected Unit/Select Unit

Syntax:

%unit?

%unit n

Description:

Function %unit? returns the currently selected unit.

Function %unit n selects the unit that will receive and process the following unit commands. Unit 0 is selected by default. If n is not a valid unit selector or the unit is not activated an error code is returned.

See also 1.1 “Units”.

Example:

```
← %unit?  
→ 1  
  
← %unit 0  
→ !0
```

%set – Set System Properties

Syntax:

%set property parameter

Description:

This function sets a system property. The property values are saved in the controller and recalled automatically after a reboot.

- **%set number-format f**: sets the format of numbers returned in command answers.
f can be one of the following:
0: *automatic*: the system uses the optimal format (fixed or scientific) for each number returned.
1: *scientific format*: numbers are returned in exponential notation, e.g. 1.4230e-03.
2: *fixed format*: numbers are returned in non-exponential notation, e.g. 0.001423.
3: *SI notation*: numbers are returned with a SI suffix, e.g. 1.423m.
 See also section 3.4 "Number Formats".
- **%set lineend-format f**: sets the format for the end of lines in answers returned from the system. f can be:
0: *CR+LF* (default): lines are terminated by a carriage-return followed by a line-feed. (ASCII 13,10)
1: *LF*: lines are terminated by a line-feed. (ASCII 10)
 When the lineend-format is changed, the answers are immediately returned with the new format, including the answer of the %set command.

Example:

```
← %set number-format 2
→ !0

← %set lineend-format 0
→ !0
```

%get – Read System Properties

Syntax:

%get property parameter

Description:

This function returns the current value of a system property.

- %get number-format: the format of numbers returned in command answers.
See %set for a list of possible values.
See also section 3.4 “Number Formats”.
- %get lineend-format: returns the format for the end of lines in answers returned from the system.
See %set for a list of possible values.

Example:

```
← %get number-format  
→ 0  
  
← %get lineend-format  
→ 1
```


%echo – Reply Text to Sender

Syntax:

```
%echo text
```

Description:

This function returns the input text unchanged to the caller.

Example:

```
← %echo hello  
→ hello
```

%save-diagnostics – Save Diagnostics Data

Syntax:

%save-diagnostics

Description:

This function saves diagnostics data as described in 2.8 "Saving Diagnostics Data".

Example:

```
← %save-diagnostics  
→ !0
```

5 SMARPOD UNITS

This section describes the command set for SmarPod units. It should be read in conjunction with the *SmarPod Programmer's Guide*. While the *Programmer's Guide* describes primarily the SmarPod C API, it contains important information about SmarPods.

5.1 SmarPod Commands

help – Show SmarPod Commands Overview

Syntax:

help

Description:

This function returns a list of all SmarPod commands with a short description.

ref?, ref – Get Referenced-Status/Start Referencing

Syntax:

```
ref?  
ref
```

Description:

Command `ref?` returns 1 if the SmarPod positioners are referenced, else 0.

Command `ref` moves all positioners to find the reference-marks. The status is not returned before the command has finished successfully or with an error, which can take some seconds. The referencing can be configured with several properties. See command `set` and chapter 5.2 "SmarPod Unit Properties" for a list of available properties. Which properties and values can be used depends on the SmarPod model.

Example:

```
← ref?  
→ 0  
← ref  
→ !0  
← ref?  
→ 1
```

confsys – Configure System for SmarPod Model

Syntax:

confsys

Description:

This command configures the MCS that controls a SmarPod. It should only be called after the SmarPod model has been changed with the system command %config-unit or if the MCS controller has been replaced. To execute this command, the unit must be deactivated.

Example:

```
← confsys  
→ !0
```

models? – List Supported SmarPod Models

Syntax:

models?

Description:

This command prints the list of supported SmarPod models. The model code (first column) can be used to configure the *model* unit option with command %config-unit.

Example:

```
← models?
→ SMARPOD MODELS:
10000    110.45 M
10001    110.45 S
10002    110.45 SC
10003     70.42 S
10004    150.57 S
10005    110.45 S-AS
10006    225.75 SD
10007     P-SLL SC
10008    110.45 S-L
10009    P-180.75 S
10010    150.57 SC

← %config-unit 0 model 10003
→ !0
```

cal – Calibrate Sensors

Syntax:

cal

Description:

This command calibrates the sensors. It is only necessary to re-calibrate the sensors if hardware has been replaced. The status is not returned before the command has finished successfully or with an error, which can take some seconds.

Example:

```
← cal  
→ !0
```

rea? – Is Pose Reachable?

Syntax:

rea? x y z rx ry rz

Description:

This command checks the reachability of the pose (x,y,z,rx,ry,rz) .

Parameters:

- x, y, z : linear translation parameters in meters.
- rx, ry, rz : rotation angles in degrees.

Example:

```
← rea? 1000 0 0 0 0 0
→ 0

← rea? 100n 250u -2.5m 0 0 5
→ 1
```


mov – Move

Syntax:

```
mov x y z rx ry rz
```

Description:

Moves the SmarPod to a new pose. If the pose is unreachable an error is returned. The command returns immediately and does not wait for the movement to be completed.

To check if the SmarPod is moving, holding or stopped the `mst?` command can be used.

Parameters:

- `x, y, z`: linear translation parameters in meters.
- `rx, ry, rz`: rotation angles in degrees. The stage is rotated by `rx` degrees around the x axis, by `ry` degrees around the y axis and by `rz` degrees around the z axis.

Example:

```
← mov 1000 0 0 0 0 0  
→ !551  
  
← mov 100n 250u -2.5m 0 0 5  
→ !0
```

stop – Stop Movement

Syntax:

stop

Description:

Stops a SmarPod movement that was started with a mov command. When sending the command once, the positioners start decelerating. To stop movements with active acceleration mode immediately, call stop twice.

Example:

```
← mov 100n 250u -2.5m 0 0 5  
→ !0  
← stop  
→ !0
```

stoh – Stop Movement And Hold Position

Syntax:

stoh [holdtime]

Description:

Stops a SmarPod movement that was started with a mov command and holds the current position. The *holdtime* parameter specifies the number of milliseconds the SmarPod will hold its position after stopping. The parameter is optional. If no holdtime is given, the position is held indefinitely.

Example:

```
← mov 100n 250u -2.5m 0 0 5  
→ !0  
← stoh  
→ !0
```

stby – Activate Standby Mode

Syntax:

stby

Description:

Puts all SmarPod positioners on standby by reducing the piezo voltage from 50 Volts (normal *stopped* status) to 0 Volts (*standby* status). After executing the command, the move status will be *moving* for a short time (about 50 to 100 milliseconds) followed by *standby*.

During the *moving* phase, the SmarPod will move slightly away from the current pose. Each positioner can move up to a few microns.

If the current status is *standby*, a subsequent call to *stop* will not change that status while *stoh* will change it to *holding*.

If the command is executed while the SmarPod is referencing or calibrating error 515 (BUSY ERROR) is returned.

Example:

```
← stby  
→ !0
```

mst? – Get Move Status

Syntax:

mst?

Description:

Returns the current move-status of the SmarPod. The following status can be returned:

- **0: stopped:** the SmarPod is not moving and the positioners are not in *holding* mode.
- **1: holding:** the SmarPod is not moving and the positioners are actively holding their positions.
- **2: moving:** the SmarPod is moving.
- **3: calibrating:** the SmarPod is calibrating the positioner sensors.
- **4: referencing:** the SmarPod is referencing the positioners.
- **5: standby:** the SmarPod positioners are in standby mode (the piezo voltage is set to 0V).

Example:

```
← mst?  
→ 1  
← mov 100n 250u -2.5m 0 0 5  
→ !0  
← mst?  
→ 2  
← stop  
→ !0  
← mst?  
→ 0
```

pos? – Get Physical Pose

Syntax:

pos?

Description:

Returns the current physical pose (x,y,z,rx,ry,rz) of the SmarPod. The pose that is returned depends on the pivot-point.

The command can be called while the SmarPod is moving.

Example:

```
← pos?  
→ 2.33333e-09 -1.73211e-09 -2.30942e-09 -1.84839e-06 -2.55378e-08 -4.31259e-06
```

piv?, piv – Get/Set Pivot-Point

Syntax:

```
piv?  
piv x y z
```

Description:

Command `piv?` returns the pivot-point, i.e. the center of rotations.

Command `piv` sets the new pivot-point (x,y,z).

The default pivot-point is (0,0,0).

Example:

```
← piv?  
→ 0 0 0  
  
← piv 50u 0 -0.1  
→ !0
```

pvm?, pvm – Get/Set Pivot-Mode

These commands have been deprecated. Use `get` and `set` instead for reading and writing properties.

Syntax:

```
pvm?  
pvm mode
```

Description:

Command `pvm?` returns the pivot-mode:

0: pivot point is relative to the stage, i.e. it moves with translation of the stage.

1: pivot point is fixed in space, i.e. it does not move with the stage.

Command `pvm` sets the pivot-mode (see above for the possible mode parameters). Parameter `mode` can be one of the values above. In relative pivot-mode the pivot point is moving with the translation of the stage. If fixed mode is set, the pivot point is fixed in space.

Example:

```
← pvm?  
→ 1  
  
← pvm 1  
→ !0
```


csy?, csy – Get/Set Coordinate System

Syntax:

```
csy?  
csy x y z rx ry rz
```

Description:

Command csy? returns the SmarPod coordinate system parameters.

Command csy changes the coordinate system.

See the SmarPod Programmers Guide for details about the coordinate system functions.

Example:

```
← csy 0 0 250u 0 7.5 -3  
→ !0  
  
← csy?  
→ 0 0 250u 0 7.5 -3
```

spz – Set Current Pose as Zero

Syntax:

spz

Description:

The command uses the current physical pose to change to coordinate system such that the current pose will become zero.

See the SmarPod Programmers Guide for details about the coordinate system functions.

Example:

```
← spz  
→ !0
```

sen?, sen – Get/Set Sensor-Mode

Syntax:

```
sen?  
sen mode
```

Description:

Command sen? returns the sensor power-mode:

- 0:** sensors are disabled.
- 1:** sensors are permanently enabled.
- 2:** sensors are enabled in power-save mode.

Command sen sets the sensor power-mode (see above for the possible mode parameters). Parameter mode can be one of the values above. With disabled sensors the SmarPod can not move.

Note: when setting the sensor-mode all positioners are stopped.

Example:

```
← sen?  
→ 1  
  
← sen 1  
→ !0
```

frq?, frq – Get/Set Maximum Frequency

Syntax:

frq?
frq frequency

Description:

Command frq? returns the current *maximum closed-loop frequency*.

Command frq sets the *maximum closed-loop frequency*.

The frequency influences the maximum velocity that can be reached by the SmarPod. If the controller needs to drive a positioner with a certain frequency and that frequency exceeds the max. frequency, the actual frequency and the velocity are capped.

Also the frequency determines the fixed movement frequency used during find-references and calibration.

Example:

```
← frq?  
→ 8000.0  
  
← frq 18.5k  
→ !0
```

vel?, vel – Get/Set Speed

Syntax:

vel?
vel speed

Description:

Command vel? returns the currently set speed.

Command vel sets the movement speed in m/s.

Example:

```
← vel 350e-6  
→ !0  
  
← vel?  
→ 350u
```

acc?, acc – Get/Set Acceleration

Syntax:

acc?
acc acceleration

Description:

Command acc? returns the currently set acceleration. If the acceleration mode is disabled, 0 is returned.

Command acc sets the movement acceleration in m/s^2 . 0 disables the acceleration mode.

Example:

```
← acc 0.000001  
→ !0  
  
← acc?  
→ 1u
```

get, set – Get/Set a SmarPod Property

Syntax:

```
get property  
set property value
```

Description:

Command `get` returns the current value of a property.

Command `set` set a property to the value given in the command. The properties are reset to their default values when the device is restarted.

See 5.2 "SmarPod Unit Properties" for a list of properties.

Example:

```
← get fref-method  
→ sequential  
  
← set fref-x-direction neg-reverse  
→ !0
```

5.2 SmarPod Unit Properties

This section lists the properties that can be written and read with the SmarPod unit set and get commands.

Property Name	Values	Description
<i>fref-method</i>		Selects the reference mark search algorithm for the <code>ref</code> command.
	default	Selects the default method for the SmarPod model.
	sequential	Selects the <i>Sequential</i> search algorithm. The positioners are referenced one at a time.
	z-safe	Selects the <i>Z-Safe</i> search algorithm. Optimizes referencing for safe movements in Z direction. First the radial positioners are referenced together so that the stage moves up or down, depending on the direction property <code>fref-z-direction</code> . Then the tangential positioners are referenced together. This method is only available for rotation-symmetric models except for SmarPods with micro-sensor positioners.
	xy-safe	Selects the <i>XY-Safe</i> search algorithm. The movement directions for the X and Y axes can be specified separately with the properties <code>fref-x-direction</code> and <code>fref-y-direction</code> . Only available for parallel SmarPods.
<i>fref-x-direction</i> <i>fref-y-direction</i> <i>fref-z-direction</i>		Properties to set the primary movement direction when executing the <code>ref</code> command.
	default	Selects the default direction for the axis.
	pos	Specifies a positive movement direction.
	neg	Specifies a negative movement direction.
	pos-reverse neg-reverse	- reverse can added to the direction for SmarPods with distance coded sensors. This will minimize to total range moved when referencing.
<i>fref-and-cal-frequency</i>		Frequency (in Hertz) used when referencing or calibrating the positioners. If this property is not set or set to 0, the frequency that was set with <code>frq</code> is used for referencing and calibration.
<i>pivot-mode</i>		Sets the behavior of the pivot point when the stage is moved linearly. Note: the pivot mode should be configured by setting this property and not with the <code>pvm</code> command anymore.
	relative	The pivot point is relative to the stage position. When the stage moves by a certain offset, the pivot moves by the same offset. This mode is useful if the SmarPod should rotate around objects that are mounted on the stage. This is the default mode.
	fixed	The pivot point is relative to the base plate of the SmarPod.

When the stage moves, the pivot point does not. This mode is useful if the SmarPod should rotate around objects that are not mounted on the stage.

5.3 SmarPod Unit Error Codes

- | | |
|-----|---|
| 1 | OTHER ERROR
An uncategorized SmarPod error has occurred. |
| 2 | SYSTEM NOT INITIALIZED ERROR
This error is returned if the initialization of the MCS controller fails. |
| 3 | NO SYSTEMS FOUND ERROR
Returned if the configured MCS controller systems could not be found. |
| 4 | INVALID PARAMETER ERROR
Returned by various functions if a parameter value is invalid and if there is no error that is more specific. |
| 5 | COMMUNICATION ERROR
Can be returned if there is a communication problem with the MCS controller. |
| 6 | UNKNOWN PROPERTY ERROR
Returned by property related function if an unknown property is passed. |
| 7 | RESOURCE TOO OLD ERROR
Returned if a resource the SmarPod (library) depends on is too old (e.g. the MCSControl library). |
| 8 | FEATURE UNAVAILABLE ERROR
Returned if a certain feature is not supported by the soft- or hardware, e.g. acceleration-control. |
| 9 | INVALID SYSTEM LOCATOR ERROR
Returned if the MCS controller locator string configured for the unit has a wrong formatting. |
| 10 | QUERYBUFFER SIZE ERROR
Returned if the user-supplied buffer is too small to hold the returned data. |
| 11 | COMMUNICATION TIMEOUT ERROR
Returned if an expected answer from the MCS controller takes too long. |
| 12 | DRIVER ERROR
Returned if a driver that is needed for communicating with the controller could not be loaded. |
| 500 | STATUS CODE UNKNOWN ERROR
Returned by status code info function if an unknown status code was passed. |
| 501 | INVALID ID ERROR
Returned if an invalid SmarPod ID was used internally. |
| 503 | HARDWARE MODEL UNKNOWN ERROR
Returned by the SmarPod initialization if called with an unknown hardware-model code. |
| 504 | WRONG COMM MODE ERROR
Returned by SmarPod initialization if the controller was initialized in synchronous mode. |
| 505 | NOT INITIALIZED ERROR
Returned by various functions if no SmarPod has been initialized for the given SmarPod ID. |
| 506 | INVALID SYSTEM ID ERROR
Returned if an invalid MCS controller ID was used internally. |
| 507 | NOT ENOUGH CHANNELS ERROR
Returned if the MCS controller has less channels than required for the configured SmarPod model. |

- 510 **SENSORS DISABLED ERROR**
Returned if the sensor mode is *disabled* when it must be enabled or in power-save mode.
- 511 **WRONG SENSOR TYPE ERROR**
No longer used. Replaced by **SYSTEM CONFIGURATION ERROR**.
- 512 **SYSTEM CONFIGURATION ERROR**
Returned if the internal MCS configuration does not match the SmarPod model passed to the initialization function.
- 513 **SENSOR NOT FOUND ERROR**
Returned if one or more sensors of the SmarPod positioners could not be detected. This error can indicate a not connected positioner or a communication problem with a positioner.
- 514 **STOPPED ERROR**
Returned if stop command is called while referencing or calibration is in progress.
- 515 **BUSY ERROR**
Returned if the SmarPod is busy with executing a command that must not be interrupted and another command is called.
- 550 **NOT REFERENCED ERROR**
Returned if the positioners have not been referenced.
- 551 **POSE UNREACHABLE ERROR**
Returned if the pose cannot be reached.
- 552 **COMMAND OVERRIDDEN ERROR**
When the software commands a movement which is then interrupted by the Hand Control Module, an error of this type is generated.
- 553 **ENDSTOP REACHED ERROR**
This error is returned if the target pose could not be reached because a mechanical end stop was detected.
- 554 **NOT STOPPED ERROR**
Returned if a command is called which requires that the SmarPod is stopped but it is moving or holding.
- 555 **COULD NOT REFERENCE ERROR**
This error is returned if referencing of the positioners could not be completed successfully.
- 556 **COULD NOT CALIBRATE ERROR**
This error is returned if calibration could not be completed successfully.

6 MCS UNITS

MCS units represent one SmarAct MCS (Modular Controller System) controller. An MCS has a fixed number of channels which are addressed by a channel index (starting at 0). Each channel can control one linear or rotary SmarAct actuator.

6.1 MCS Commands

This chapter describes the command set for MCS units. It should be read in conjunction with the *MCS Programmers Guide*. While the *Programmers Guide* describes primarily the MCS C API, it contains important information about MCS.

help – Show MCS Commands Overview

Syntax:

```
help
```

Description:

This function returns a list of all MCS commands with a short description.

nch? – Get Number Of Channels

Syntax:

nch?

Description:

This function returns the number of channels of the MCS.

Example:

```
← nch?  
→ 3
```

sta? – Get Channel Status

Syntax:

sta? c

Description:

This function returns the current status of channel c as a number. For a list of channel status codes see section 6.3 MCS Unit Channel Status Codes.

Example:

```
← sta? 0  
→ 3      channel 0 is in HOLDING state
```

sty – Set Sensor Type

Syntax:

sty c type

Description:

This function sets the sensor type of channel c. When using positioners with integrated sensors, this command may be used to tell a channel what type of positioner is connected. The type affects position calculation and commands that may be issued for a channel. For example, a channel that is configured as rotary will not accept commands for linear movements.

Note that each channel stores this setting to non-volatile memory. Consequently, there is no need to issue this command on every initialization. If the sensor type of a channel is changed, you must calibrate the channel to ensure proper operation of the sensor. If this command is issued, the positioner is implicitly stopped.

Example:

```
← sty 2 1  
→ !0
```

sty? – Get Sensor Type

Syntax:

sty? c

Description:

This function returns the sensor type of channel c. See sty for more details.

Example:

```
← sty? 2  
→ 1
```


cal – Calibrate Sensor

Syntax:

cal c

Description:

This command calibrates the sensor of channel c.

It may be used to increase the accuracy of the position calculation. It is only executable by a positioner that has a sensor attached to it. The sensor must also be enabled or in power save mode (→ sen). If this is not the case the channel will return an error. This command should be called once for each channel if the mechanical setup changes (different positioners connected to different channels). The calibration data will be saved to nonvolatile memory. If the mechanical setup is unchanged, it is not necessary to send this command on each initialization, but newly connected positioners have to be calibrated in order to ensure proper operation.

During the calibration the positioner will perform a movement of up to several mm. You must ensure, that the command is not executed while the positioner is near a mechanical end stop. Otherwise the calibration might fail and lead to unexpected behavior when executing closed-loop commands. As a safety precaution, also make sure that the positioner has enough freedom to move without damaging other equipment. The calibration takes a few seconds to complete. During this time the positioner will report a status code of 6 (CALIBRATING).

Positioners that are referenced via a mechanical end stop are moved to the end stop as part of the calibration routine. Which end stop is used for referencing can be configured with the sfd command. Note that when changing the safe direction the end stop must be calibrated again for proper operation.

The command returns immediately, while the calibration can take several seconds. The calibration process can be observed by checking the channel status with the sta? command. While the channel is calibrating, the status is 6 (CALIBRATING). When calibration has finished, the status is 0 (STOPPED).

Example:

```
← cal 1
→ !0
← sta? 1
→ 6
. . . (wait until channel activity stops)
← sta? 1
→ 0
```

ref?, ref – Get Referenced Status/Start Referencing

Syntax:

```
ref? c
ref c d z
```

Description:

Command `ref?` returns 1 if channel `c` is referenced, else 0.

Command `ref` starts referencing channel 0 in direction `d`, which can be `f` (forward direction) or `b` (backward direction). Note that this parameter is ignored for sensor types that are referenced via a mechanical end stop. If the auto-zero parameter `z` is 1, the position will be set to 0 at the reference-mark if the referencing completes successfully. When referencing has finished successfully, the MCS will hold the position actively for the hold-time that has been set with command `htm`.

The command returns immediately, while the referencing process can take several seconds. The process can be observed by checking the channel status with the `sta?` command. While the channel references, the status is 7 (FINDING REF). When referencing has finished, the status will be either 0 (STOPPED) or 3 (HOLDING). The referencing was successful if `ref?` returns 1.

Example:

```
← ref? 0
→ 0          channel 0 not referenced
← ref 0 f 1
→ !0        referencing of channel 0 has started
← sta? 1
→ 7          channel 0 is referencing

... (wait until channel activity stops)

← sta? 1
→ 0          channel 0 is in STOPPED state, referencing has stopped, but was it successful?...
← ref? 0
→ 1          ... yes it was. Channel 0 is now referenced.
← pos? 0
→ 0          since ref was called with the auto-zero = 1, the current position is (close to) 0
```

htm – Set Hold Time

Syntax:

htm c t

Description:

This function returns the hold time of channel c. This is the time a channel will actively hold the target position after a successful closed-loop move. The time t must be specified in milliseconds. Values of 0 to 60000 are possible. When a hold time of 60000 is set, the channel will hold the position indefinitely.

Example:

```
← htm 2 2500  
→ !0
```

frq – Set Frequency

Syntax:

frq c f

Description:

This function sets the frequency parameter of channel c. For positioners that have a sensor installed, this function may be used to define the maximum frequency that the positioners are driven with when issuing closed-loop movement commands. Frequency values between 50 and 18500 are possible.

Example:

```
← frq 0 18500  
→ !0
```

vel?, vel – Get/Set Closed-Loop Speed

Syntax:

```
vel? c  
vel c s
```

Description:

Function vel? returns the closed-loop speed of channel c.

Function vel set the closed-loop speed of channel c to s in meter per second.

Example:

```
← vel? 0  
→ 2.5e-3  
← vel 0 1.5m  
→ !0  
← vel? 0  
→ 1.5e-3
```

sfd – Set Safe Direction

Syntax:

```
sfd c d
```

Description:

This function sets the safe direction for channel c to d, which can be f (forward) or b (backward).

Example:

```
← sfd 0 f  
→ !0
```

sen?, sen – Get/Set Sensor Mode

Syntax:

```
sen?  
sen m
```

Description:

Command sen? returns the current mode of the sensors connected to the MCS.

Command sen set the sensor mode of all channels to m, which can be 0 (disabled), 1 (enabled), 2 (power-save mode). It effectively turns the power supply of the sensors on or off. End effector channels are not affected by this function. If this command is issued, all positioner channels of the system are implicitly stopped. This setting is stored to non-volatile memory immediately and need not be configured on every power-up.

Example:

```
← sen?  
→ 1  
← pos? 0  
→ 1.321e-4  
← sen 0  
→ !0  
← sen?  
→ 0  
← pos? 0  
→ !140 (error: sensor disabled)
```

mpa, mpr – Move To Absolute/Relative Position

Syntax:

```
mpa c p  
mpr c r
```

Description:

Command `mpa` moves the linear positioner connected to channel `c` to the absolute position `p` (in meters).

Command `mpr` moves a linear positioner relative to its current position by distance `r` (in meters).

When the channel has reached the target position, the MCS will hold the target position actively for the hold-time that has been set with command `htm`.

The command returns immediately, while the movement can take much longer. The movement can be observed by checking the channel status with the `sta?` command. While the channel is moving, the status is 4 (TARGET). When moving has finished, the status is 0 (STOPPED) or 3 (HOLDING).

Example:

```
← mpa 0 250u  
→ !0  
← mpr 0 1.5m  
→ !0
```


maa, mar – Move To Absolute/Relative Angle

Syntax:

```
maa c a r  
mar c d r
```

Description:

Command maa moves the rotary positioner connected to channel c to the absolute angle a (in degrees) and r revolutions. Parameter a can be [0,360] and r can be [-32768...32767].

Command mar moves a rotary positioner relative to its current angle by delta angle d (in degrees) and by a revolution difference r. Parameter d can be [-360,360] and r can be [-32768...32767].

When the channel has reached the target angle, the MCS will hold the target angle actively for the hold-time that has been set with command htm.

The command returns immediately, while the movement can take much longer. The movement can be observed by checking the channel status with the sta? command. While the channel is moving, the status is 4 (TARGET). When moving has finished, the status is 0 (STOPPED) or 3 (HOLDING).

Example:

```
← maa 0 270 -1  
→ !0  
  
← mar 0 90 1  
→ !0
```

mst – Step-Move Positioner

Syntax:

mst c s a f

Description:

The command moves the positioner at channel *c* open-loop a number of piezo-motor steps *s* with amplitude *a* and frequency *f*.

Parameter steps *s* can be ± 30000 , amplitude *a* can be [0...4095] and frequency *f* [1...18500].

The command returns immediately, while the movement can take much longer. The movement can be observed by checking the channel status with the `sta?` command. While the channel is moving, the status is 1 (STEPPING). When stepping has finished, the status is 0 (STOPPED).

Example:

```
← mst 0 1500 4095 12000  
→ !0
```

stop – Stop One Or All Channels

Syntax:

```
stop c  
stop
```

Description:

The command stops either one channel, if a channel parameter is c included, or all channels at once.

The resulting channel status will be 0 (STOPPED).

Example:

```
← stop 2  
→ !0
```

pos? – Get Position

Syntax:

pos? c

Description:

The command returns the position of a linear channel in meters.

Example:

```
← pos? 2  
→ -1.230002e-3
```

ang? – Get Angle

Syntax:

ang? c

Description:

The command returns the angle and revolution of a rotary channel in degrees. See `maa` for more details.

Example:

```
← ang? 2  
→ 90 -2
```

6.2 MCS Unit Command Error Codes

- 1 **INITIALIZATION ERROR**
An error occurred while initializing the library. All systems should be disconnected and reset before the next attempt is made.
- 2 **NOT INITIALIZED ERROR**
A function call has been made for an uninitialized system.
- 3 **NO SYSTEMS FOUND ERROR**
May occur at initialization if no Modular Control Systems have been detected on the PC system.
- 4 **TOO MANY SYSTEMS ERROR**
The number of allowed systems connected to the PC is limited to 32. If you have more connected, disconnect some.
- 5 **INVALID SYSTEM INDEX ERROR**
An invalid system index has been passed to a function. The system index parameter of various functions is zero based. If N is the number of acquired systems, then the valid range for the system index is 0..(N-1).
- 6 **INVALID CHANNEL INDEX ERROR**
An invalid channel index has been passed to a function. The channel index parameter of various functions is zero based. If N is the number of channels available on a system, then the valid range for the channel index is 0..(N-1).
- 7 **TRANSMIT ERROR**
An error occurred while sending data to the hardware. The system should be reset.
- 8 **WRITE ERROR**
An error occurred while sending data to the hardware. The system should be reset.
- 9 **INVALID PARAMETER ERROR**
An invalid parameter has been passed to a function. Check the function documentation for valid ranges.
- 10 **READ ERROR**
An error occurred while receiving data from the hardware. The system should be reset.
- 12 **INTERNAL ERROR**
An internal communication error occurred. The system should be reset.
- 13 **PROTOCOL ERROR**
An internal protocol error occurred. The system should be reset.
- 15 **TIMEOUT ERROR**
The hardware did not respond. Make sure that all cables are connected properly and reset the system.
- 19 **WRONG CHANNEL TYPE ERROR**
A command could not be started because the channel configuration does not allow to execute this command.
- 129 **NO SENSOR PRESENT ERROR**
This error occurs if a function was called that requires sensor feedback, but the addressed positioner has none attached.

- 130 **AMPLITUDE TOO LOW ERROR**
The amplitude parameter that was given is too low.
- 131 **AMPLITUDE TOO HIGH ERROR**
The amplitude parameter that was given is too high.
- 132 **FREQUENCY TOO LOW ERROR**
The frequency parameter that was given is too low.
- 133 **FREQUENCY TOO HIGH ERROR**
The frequency parameter that was given is too high.
- 135 **SCAN TARGET TOO HIGH ERROR**
The target position for a scanning movement that was given is too high.
- 136 **SCAN SPEED TOO LOW ERROR**
The scan speed parameter that was given for a scan movement command is too low.
- 137 **SCAN SPEED TOO HIGH ERROR**
The scan speed parameter that was given for a scan movement command is too high.
- 140 **SENSOR DISABLED ERROR**
This error occurs if an addressed positioner has a sensor attached, but it is disabled.
- 141 **COMMAND OVERRIDEN ERROR**
This error is only generated in the asynchronous communication mode. When the software commands a movement which is then interrupted by the Hand Control Module, an error of this type is generated.
- 142 **END STOP REACHED ERROR**
This error is generated by a positioner channel in asynchronous mode if the target position of a closed-loop command could not be reached because a mechanical end stop was detected. After this error the positioner will have the STOPPED STATUS status code.
- 143 **WRONG SENSOR TYPE ERROR**
This error occurs if a closed-loop command does not match the sensor type that is currently configured for the addressed channel.
- 144 **COULD NOT FIND REF ERROR**
This error is generated in asynchronous mode if the search for a reference mark was aborted.
- 145 **WRONG END EFFECTOR TYPE ERROR**
This error occurs if a command does not match the end effector type that is currently configured for the addressed channel.
- 146 **MOVEMENT LOCKED ERROR**
Generated either if a movement command was aborted due to an emergency stop or a movement command was issued while the channel is in the locked state.
- 147 **RANGE LIMIT REACHED ERROR**
If a range limit is defined and the positioner is about to move beyond this limit, then the positioner will stop and report this error. After this error the positioner will have the STOPPED STATUS status code.
- 148 **PHYSICAL POSITION UNKNOWN ERROR**
A range limit is only allowed to be defined if the positioner “knows” its physical position. If this is

not the case, the functions `SetPositionLimit X` and `SetAngleLimit X` will return this error code.

149 OUTPUT BUFFER OVERFLOW ERROR

When using buffered output in asynchronous communication mode, this error is returned if too many commands were accumulated in the output buffer.

150 COMMAND NOT PROCESSABLE ERROR

This error is generated if a command is sent to a channel when it is in a state where the command cannot be processed. For example, to change the sensor type of a channel the addressed channel must be completely stopped. In this case send a stop command before changing the type.

151 WAITING FOR TRIGGER ERROR

If there is at least one command queued in the command queue then you may only append more commands (if the queue is not full), but you may not issue movement commands for immediate execution. Doing so will generate this error.

152 COMMAND NOT TRIGGERABLE ERROR

Commands that cannot be triggered will generate this error.

153 COMMAND QUEUE FULL ERROR

This error is generated if you attempt to append more commands to the command queue, but the queue cannot hold anymore commands.

154 INVALID COMPONENT ERROR

Indicates that a component was selected that does not exist.

155 INVALID SUB COMPONENT ERROR

Indicates that a sub component was selected that does not exist.

156 INVALID PROPERTY ERROR

Indicates that the selected component does not have the selected property.

157 PERMISSION DENIED ERROR

This error is generated when you call a functionality which is not unlocked for the system (e.g. Low Vibration Mode).

161 INCOMPLETE PACKET ERROR

This error is generated by the hardware if a packet was not received completely and thus a timeout occurs.

164 RX BUFFER OVERFLOW ERROR

The system could not process all incoming command packets and an internal buffer overflow occurs. This indicates to a potential misuse of the programming API. Too many commands were send without reading the returned answer packets. The system went to an error state where further communication is blocked until the system is restarted.

240 UNKNOWN COMMAND ERROR

This error occurs if the library sends a command that is not supported by the system it is sent to. This may be the case when using a newer library with an older firmware version. Please be sure only to use the library that is shipped with your system to avoid compatibility problems.

255 OTHER ERROR

An error that can't be otherwise categorized.

6.3 MCS Unit Channel Status Codes

MCS channels have a status from the following table. The current channel status can be checked with the `MCS sta?` command.

Status	Name	Description
0	STOPPED	The positioner or end effector is currently not performing active movement.
1	STEPPING	The positioner is currently performing an open-loop movement.
2	SCANNING	The positioner is currently performing a scanning movement.
3	HOLDING	The positioner or end effector is holding its current target.
4	TARGET	The positioner or end effector is currently performing a closed-loop movement.
5	MOVE DELAY	The positioner is currently waiting for the sensors to power-up before executing the movement command. This status may be returned if the the sensors are operated in power save mode.
6	CALIBRATING	The positioner or end effector is busy calibrating its sensor.
7	FINDING REF	The positioner is moving to find the reference mark.
8	OPENING STATUS	The end effector (gripper) is closing or opening its jaws.

7 APPENDIX

7.1 Example Command Sequence

← %set number-format 3	select the number format for answers to <i>SI format</i>
→ !0	ok
← %unit 0	select unit 0 (in this example it is a SmarPod unit)
→ !0	
← ref?	check if the SmarPod is referenced
→ 0	not referenced...
← set fref-and-cal-frequency 8k	...set frequency for referencing movements...
→ !0	
← set fref-method z-safe	...set method used for referencing
→ !0	
← ref	...and start referencing
→ !0	status is returned when referencing is over (can take some seconds)
← frq 18.5k	set the maximum frequency for normal movements
→ !0	
← vel 200u	set the speed to 200 µm/s
→ !0	
← mov 0 0 650 0 0 0	move to z=650 (meters!)
→ !551	error: this pose is not reachable, of course
← %code? 551	what does SmarPod error code 551 mean?
→ pose unreachable	
← mov 0 0 650u 0 0 0	we wanted microns, not meters: move to z=650µm
→ !0	status is returned immediately, the movement might be in progress...
← mst?	check the move-status
→ 2	moving
← pos?	get the current pose...
→ 0 0 200u 0 0 0	
← mst?	check the move-status
→ 2	still moving
← mst?	check the move-status
→ 1	move-status is <i>holding</i> : move completed
← pos?	get the current pose
→ 0 0 650u 0 0 0	
← mov 0 0 0 0 0 0	move back to zero
→ !0	
← stop	stop the movement
→ !0	
← mst?	check the move-status
→ 0	move-status is <i>stopped</i> : the positioners are not holding their positions

7.2 System Error Codes

0	ok
10001	other error
10002	syntax error
10003	unknown command
10004	invalid parameter
10005	feature unavailable
10006	could not connect: maximum number of network connections reached
10007	unit-server inactive
10100	unit selection invalid
10101	unit not activated
10102	unit activated
10103	unit activate failed
10104	unit deactivate failed