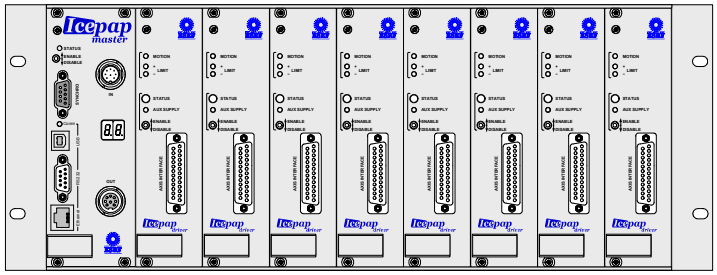
ESRF ISG

ESRF - Instrument Support Group

Icepap

Intelligent Controller for Positioning Applications

User Manual



DRAFT IN CONSTRUCTION

version 0.0c / 04.12.2009

Date	Version	Comments
04/12/2009	0.0c	Draft in construction

CONTENTS

MANUAL ORGANIZATION

This manual presents ….

Section 1 gives a brief overview of … The description is made in general terms and specific technical details are minimised.

Section 2 describes …. The connectors and signals functions of both front and rear panels are detailed.

Section 3 is dedicated to … in a *real world* setup.

Section 4 covers the available commands to communicate with IcePAP systems.

Section 5 is intended to describe the programming aspects of ….

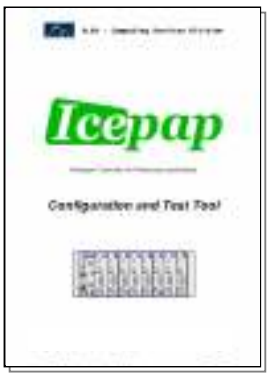
Related Documentation

- *IcePAP Hardware Manual*

Presents in detail the components and functionality of the IcePAP system and provides a complete connector description.

- *IcePAP Configuration and Test Tool*

Describes the GUI tool used for driver configuration and testing.



1. INSTALLATION

1.1. System overview

IcePAP is a motor control system developed at the ESRF and optimised for high resolution position applications. An IcePAP system may drive up to 128 axes and integrates both control features, like trajectory generation, and the motor power management. Although motor control in IcePAP is axis-oriented, it includes system resources that allow the execution of synchronous multi-axis movements. In addition, all the position information signals are driven through internal multiplexers and can be sent to external devices to properly synchronise data acquisition during motion.

Besides high performance, IcePAP is fully software configurable and provides exhaustive diagnostic capabilities. Most of the functionality relies on programmable components what opens the possibility of adding new features by means of firmware upgrade.

Components

The IcePAP system is organised in racks. The mechanical support of each rack is provided by a 19" 3U crate that includes the power supply and an interconnection backplane with 9 slots.
The leftmost slot is wider than the others and must be always equipped with a controller board. The 8 remaining slots may be equipped with 1 to 8 driver boards. Each driver board can operate a motorised axis.
The unused slots must be covered with blank front panel plates to avoid accidental access to internal parts with electrical power.

Figure 1 depicts an IcePAP crate populated with 5 driver boards.

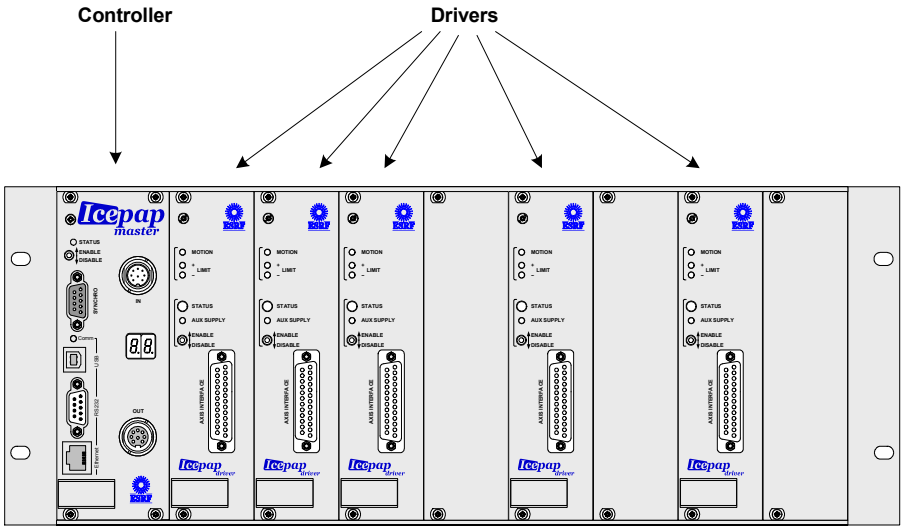


Figure 1: Example of a partially equipped IcePAP rack

Several racks can be connected to form a single multirack IcePAP system. Each rack must be identified with a different number (0 to 15) that is visualised in a two-digit display at the front panel of the controller board.
From the point of view of hardware implementation there are two types of controller boards: MASTER and SLAVE. MASTER controllers have additional hardware resources such as a communication processor and certain connectors that are not available in the SLAVE

controllers. A more complete description of the IcePAP system and its hardware resources can be found in the *IcePAP Hardware Manual*.

An IcePAP system must always include a rack 0. The controller in the rack 0 must always be a MASTER board that plays the role of system master and takes care of communications and system management.
In multirack systems the other controllers operate always as slave boards regardless of whether or not they are physically MASTER or SLAVE controllers. A MASTER board can be installed in any rack and operate as slave controller, but a SLAVE board can never operate as system master controller and must never be inserted in rack 0.
Single rack systems must therefore always be identified with rack number 0 and include a MASTER board as controller.

1.2. Hardware connections and configuration

1.2.1. Rack number

Rotary switch (hexadecimal).
Controller must be extracted and rack power switched off (switch at the back).
Once the rack number is properly set, the controller board can be plugged in the leftmost slot.

1.2.2. Board installation (controllers and drivers)

No hardware configuration required. Driver configuration is fully implemented by software commands. Mandatory to use the configuration tool.
When a driver board is installed the first time or moved from one slot to a different one, the board becomes not active (see ...)

1.2.3. Rack links and termination

Multi-axis system needs rack links
lengths, wiring.
Bus terminator.

1.2.4. Rack disable

Each rack includes a disable connector at the rear panel that allows to disable remotely the motor power of all the driver boards in the rack.
If not used ([what happens?])

1.2.5. Communication links

The available communication interfaces are RS232 and Ethernet. Note that the USB interface is not available in the current version. The communication interface connectors are accessible at the front panel of the system master controller. See details in the *IcePAP Hardware Manual*.

RS232: This is an asynchronous serial port that should be configured as follows:

Baudrate	???
Parity	NO
Stop bits	1

Table 1. Serial Line (RS232) settings

Ethernet:
[how to set the IP addres???] [protocol TCP, ports to use, ...]

1.2.6. Motor and encoder connection

As described in the Hardware Manual.
Connecting the axis disable line is mandatory
Proper grounding and shielding.

1.2.7. Ventilation

IcePAP racks do not include internal fans of other method for forced ventilation. External ventilation may be necessary .

It is recommended that …
[At least the case of installation in closed 19" cabinets.]

1.3. Installation tips

- It is always useful to check ….
- Use ...

2. OPERATION INSTRUCTIONS

This section deals with ...

2.1. IcePAP concepts

2.1.1. Systems and boards

As explained in 1.1, an IcePAP system may include a variable number of driver boards organised in several racks. Every rack in the system must have an unique identification number that is displayed in the front panel of the rack controller board. The controller board in rack number 0 acts as system master controller and manages communication and system functionality. The communication cable (RS232 or Ethernet) must be plugged into the corresponding connector of the system master controller.

Every board in the system has an unique address *A* that is a decimal number formed as $A = 10 \times R + S$, where *R* is the rack number and *S* is the slot occupied by the board in the rack. Rack controllers always occupy the slot 0, while driver boards are installed in the slots 1 to 8. With this convention the last decimal digit of the board address easily identifies if the board is a controller or a driver and its slot number. The address 0 is always assigned to the system master.

The system master receives and processes ASCII commands from the host computer. Commands that request an answer from an IcePAP module always start by a question mark character ('?') and are often called "queries" in this document.
There are two types of commands and queries: if the received command includes a prefix with a numeric address, it is treated as a board command or board query and is dispatched to and executed by the corresponding board. Board commands can be addressed to both controllers and drivers. If the command does not include an address prefix, it is treated as a system command or system query that is directly processed and executed by the system master controller.
It is also possible to address board commands and queries to all the boards in the system by using a broadcast mechanism. See chapter 4 for more details on the communication protocol and the command format.

When a system command and a board command have analogous functionality, they usually share the same name. However system and board commands are processed in a different way and have slightly different effect and therefore should not be considered as being the same command. As presented in chapter 5, system commands and board commands are always considered as two different sets.
Some board commands are not implemented in rack controllers and can only be executed by driver boards. Note also that the system master controller executes both system commands as well as board commands sent to the address 0.

At any time the host computer may obtain information about the current configuration of an IcePAP system by issuing ?SYSSTAT system queries. This query reports information about the racks connected in the system, the driver boards plugged in each rack and if the plugged boards are responsive or not.

Another useful system query is ?MODE that returns the current system mode. In normal operation conditions an IcePAP system must be always in OPER mode (see ?MODE system query). Other system modes are useful only for maintenance interventions such as firmware reprogramming or factory testing. Every board can also return its individual mode by the ?MODE board query. The only cases in which the board mode may be reported as different from the system mode are either when a driver is being configured (CONFIG mode), or when there is an unrecoverable hardware failure that switches a driver board to FAIL mode.

2.1.2. Configuring drivers

what is in a driver (indexer + power stage). Concept of indexer

Possibility of internal or external indexer/power stage
Power on/off issues (see section 2.1.3)

By software commands
Stored in non-volatile memory.
The detailed description of the configuration parameters is presented in chapter 3.
But use the configuration Tool

2.1.3. Enabling and disabling axes

axis activation: ?ACTIVE and ACTIVE commands.
Alarms and warnings

2.1.4. Motor types

motor configuration (makes sense here?)

- Dealing with linear and rotary motors

2.1.5. Encoders

functional encoders (target and shaft encoders)
physical and functional encoders

2.1.6. Position resolution

- position resolution:
- axis position concept

2.1.7. Closed loop operation

2.1.8. Advanced functionality

- position control

- Homing procedures

- I/O signals

2.2. Moving motors

2.2.1. Basic movements

In the most usual case, the various IcePAP axes are operated as independent channels
MOVE, RMOVE, JOG
?FSTATUS, ?STATUS

2.2.2. Homing and search sequences

Each driver has the possibility to latch all of its position sources at a specified event. This capability is used in two built-in commands that start a sequence that will search the position of an external reference signal, thus allowing to find an absolute position of the axis..

2.2.3. Multiaxis and group movements

Multiaxis motion commands. It is possible to move several axes by using the system motion commands MOVE, RMOVE, JOG and HOME. The multiaxis versions of these commands first check that all the axes in the parameter list can be moved as instructed. If there are invalid parameters or any of the movements cannot be started, the command fails and returns an error. Only once the initial check is successful, all the axes start their movements simultaneously.

The progress of the movement can then be followed by monitoring the status register of the individual axes. It is possible however to read the status of a set of axes by using the ?FSTATUS system query.

Axis groups. Once a multiaxis movement is started, by default every axis is managed as independent from the others. The motion of a given axis finishes either when the motion sequence is completed, when the driver receives a stop command or if a limit switch, an alarm or an error condition is found.
This default behaviour can be changed if the GROUP keyword is used as a flag in the multiaxis command. In that case all the axes included in the same multilink command are not only started simultaneously but are also internally linked as an active group. Whenever any of the axes in an active group is stopped by a STOP command, a limit switch or an alarm condition, all the other axes in the group are forced to stop immediately.

2.3. Diagnostics

2.3.1. Status registers

The status of each IcePAP board is compiled in a 32 bit register that can be read by the ?FSTATUS, ?STATUS and ?VSTATUS queries.
The ?FSTATUS query returns the board status that is stored in the master controller and therefore present the shortest response latency. It is the recommended query for intensive polling applications.
The ?STATUS query returns the status register read directly from the individual board and therefore guarantees the most updated values.
The ?VSTATUS query reports the board status information in a verbose form that is intended for diagnostics and assistance to application programmers.

The status information consists of a number of status bits and fields that are summarised in Table 2 and described below:

PRESENT: This bit is set when the board is found to be physically present in the system. Note that the ?FSTATUS query may return the status of non present boards.

ALIVE: This bit is set when the board communicates by the internal bus and responds to commands and queries.

MODE: This field represents the current functional mode of the board. See the ?MODE board query for more details.

DISABLE: This field is set to a non zero value if the motor power is disabled. The power can be enabled and disabled by the software POWER command or by the front panel switches. It can also be permanently disabled if the axis is configured as not active, by one of the external disable signals or if an alarm condition happens. In case of alarm conditions, the *STOPCODE* field provides more details about the reason of the alarm. See 2.3.3 for more information about alarm sources.

INDEXER: This field indicates if the axis trajectories are generated by the internal indexer, by an in-system indexer signal through the backplane or by and external indexer connected to one of the input position signals InPos or EncIn.

READY: This bit is set when the board is ready to accept new motion commands. It must be checked before starting a new movement. If it is not set, the axis is still busy in a operation such as a point to point movement, a settling phase or in a homing sequence.

MOVING: This bit indicates that the axis is in motion. It must be used only for informative purposes and not to decide when the motion is completed and when the axis is able to accept new motion commands. Use the *READY* bit instead.

SETTLING: In closed loop, this bit is set during the settling phase and is cleared once the settling condition is met. See 2.1.7 for more details about closed loop operation.

OUTOFWIN: This bit is set if the axis position is out of the target position window. See 2.1.7 for more details about closed loop operation.

WARNING: This bit is set if a warning condition has been met. See 2.3.2 for more information about possible source of warnings.

STOPCODE: This field is 0 if the last motion command completed successfully with no interruption. If the movement was interrupted or not even started, a value from 1 to 6 reports the reason. If the power is disabled because the board went into alarm state, a value from 8 to 15 indicates the specific alarm condition. Note in this last case, the *DISABLE* field must also signal that an alarm condition was met.

LIMIT+, LIMIT-: These bits report the actual logic level of the Limit+ and Limit- signals connected at the rear driver connectors.

HSIGNAL: This bit reports the logic level of the homing reference signal. Note that the homing reference signal is selected by the HOMESRC configuration parameter and it may be different from the driver Home signal connected at the rear panel.

VERSERR: This bit is set to if the version of the board firmware is not consistent with the firmware version of the master controller.

INFO: This field provides a progress monitor during the board programming procedures.

		DRIVER Status	*CONTROLLER Status*
Bit #	**name**	**value = description**	**value = description**
0	PRESENT	1 = driver present	1 = controller present
1	ALIVE	1 = board responsive	1 = board responsive
2-3	MODE	0 = OPER 1 = PROG 2 = TEST 3 = FAIL	0 = OPER 1 = PROG 2 = TEST 3 = FAIL
4-6	DISABLE	0 = power enabled 1 = axis configured as not active 2 = alarm condition 3 = remote rack disable input signal 4 = local rack disable switch 5 = remote axis disable input signal 6 = local axis disable switch 7 = software disable	0 = power enabled 1 = n/a 2 = alarm condition 3 = remote rack disable input signal 4 = local rack disable switch 5 = n/a 6 = n/a 7 = software disable
7-8	INDEXER	0 = internal indexer 1 = in-system indexer 2 = external indexer 3 = n/a	0 = internal indexer 1 = n/a 2 = n/a (status of multiplexer?) 3 = n/a
9	READY	1 = ready to move	1 = ready to move
10	MOVING	1 = axis moving	1 = virtual axis moving
11	SETTLING	1 = closed loop in settling phase	n/a
12	OUTOFWIN	1 = axis out of settling window	n/a
13	WARNING	1 = warning condition	n/a
14-17	STOPCODE	0 = end of movement 1 = STOP 2 = ABORT 3 = LIMIT+ reached 4 = LIMIT- reached 5 = settling timeout 6 = axis disabled (no alarm condition) 7 = n/a 8 = internal failure 9 = motor failure 10 = power overload 11 = driver overheating 12 = close loop error 13 = control encoder error 14 = n/a 15 = external alarm	0 = end of movement 1 = STOP 2 = ABORT 3 = n/a 4 = n/a 5 = n/a 6 = n/a 7 = n/a 8 = internal failure 9 = n/a 10 = n/a 11 = n/a 12 = n/a 13 = n/a 14 = n/a 15 = external alarm
18	LIMITPOS	current logic value of the Limit+ signal	n/a
19	LIMITNEG	current logic value of the Limit- signal	n/a
20	HSIGNAL	current value of the homing ref. signal	n/a
21	5VPOWER	1 = Aux power supply on	n/a
22	VERSERR	1 = inconsistency in firmware versions	1 = inconsistency in firmware versions
23	POWERON	1 = Motor power on	n/a
24-31	INFO	In PROG mode: programming phase In OPER mode: master indexer	In PROG mode: programming phase

Table 2. Driver and controller board status registers

2.3.2. Warnings

Use the ?WARNING query

...

2.3.3. Alarms

...

2.4. Firmware reprogramming

2.5. Usage tips

Refer to Section 5.1 for all the commands mentioned below.

- It is quite common to have a situation where one wants to change an incremental encoder direction sign. This can be done easily by using the *INV* keyword in the correspondent channel configuration. Moreover, the *INV* keyword can be also used to change the polarity of the input signals (see the *CHCFG* command).

- It is always useful to check the …. The command allows .

3. DRIVER CONFIGURATION

Commands: CONFIG, ?CONFIG, CFG, ?CFG, ?CFGINFO
Configuration parameters cannot be changed in normal operation mode.

3.1. Axis configuration

3.1.1. Axis activation and protection

ACTIVE parameter and ?ACTIVE and ACTIVE commands.
A configuration parameter for selection of additional levels of protection (PROTLEVEL) is foreseen but currently not implemented.

3.1.2. Axis name

The name of the axis is a character string that is stored in the non-volatile memory and is only used for identification purposes. The axis name is not a configuration parameter and can be in principle changed during operation by the host computer. It is however possible to forbid changes of the axis name in operation mode by using the NAMELOCK configuration flag.

3.1.3. Default indexer

In most of the cases a driver will be operated by using its internal indexer to generate the motion trajectory, velocity profile, etc. It is possible however to bypass the internal indexer and drive the motor using pulses coming from an external device such as another IcePAP driver or a multiaxis controller. The actual indexer used by a particular driver can be changed during operation by a user command but the default indexer that is used after power on or after driver reset must be selected by the parameter INDEXER. As mentioned above in most of the cases the default indexer will be set to INTERNAL.

3.1.4. Position source and resolution

The position resolution of a given IcePAP axis can be defined arbitrarily and it is independent of the motor and encoders associated to that axis. Although the IcePAP resolution can be defined higher than the actual mechanical resolution of the axis, the use of excessively high values is discouraged because it is both useless in practice and can be a potential source of problems such as producing position overflows that may be difficult to track and debug. IcePAP drivers store axis positions internally with very high numeric resolution, but the position values are converted to 32-bit signed integer values when transferred across the system or to the host computer.

The resolution of an axis is defined by the configuration parameters ANSTEP and ANTURN as the integer number of steps (or position counts) that corresponds to a given integer number of motor turns. In the case of dealing with linear motors instead of rotary motors, the number of motor turns must be replaced by the number of reference displacement intervals, where a reference displacement interval must be the same linear displacement used to configure the number of pole pairs of the motor (see ...).
In most of the cases it is a good practice to set ANTURN to 1 and set ANSTEP to the desired number or steps per motor turn or per unit of reference displacement in the case of linear motors.

The axis resolution defines the *axis step* unit that is used to report axis positions and position errors as well as to define parameters such as velocities. The internal indexer always operate *axis steps* units and all displacements and position values in movement commands are expressed in such units.
The nominal axis position can be selected to be taken from the current indexer value or from the one of the encoders associated to the axis. This behaviour is configured by the POSSRC parameter.

The ?POS query may be used to return different position values from the axis al or

The position of axis
POSSRC

3.2. Motor configuration

3.2.1. Motor types

Supported types

3.3. Advanced configuration

3.3.1. Homing configuration

Homing configuration parameters

3.4. Configuration parameters

ACTIVE { NO | YES } — **Axis enable/disable flag**

This parameter marks a driver board as active or not. A “non active” driver is disabled, cannot be used to drive motors and rejects most of the power and motion related commands. The ACTIVE parameter does not reflects necessarily the actual state of a driver board that can become “not active” (functionally disabled) if is moved to a different IcePAP system. See xxx for more information.

PROTLEVEL <integer> — **Protection level**

This value is not actually used by the icePAP drivers. It is provided as a way to store locally information about the level of protection that must be applied to the corresponding axis. This value is available to be used by the application software.

NAMELOCK { NO | YES } — **Axis name lock**

If this flag is set to NO, the use of the NAME command to change the name of the driver board is not allowed.

POWERON { NO | YES } — **Auto power on**

This flag instructs the driver board to switch on the motor power immediately after board initialisation. The flag has effect only if the driver is active.

MOTPHASES { 1 | 2 | 3 } — **Number of electrical phases**
MOTPOLES <integer> — **Number of pole pairs**

Configure the number of electrical phases and pole pairs of the motor. In case of rotary motors, the number of pole pairs corresponds to the number of electrical periods per motor turn. For instance, this number is 50 for a standard 200 full steps per turn stepper motor.
In case of linear motors, the number of pole pairs corresponds to the number of electrical periods for a certain given displacement distance. Such a distance is somehow arbitrary and can be chosen according to the user convenience, but will be adopted as the effective “*motor revolution*” for all internal calculations. All the configuration parameters that refer to motor turns will actually apply to such a reference linear displacement.

MOTSENSE { NORMAL | INVERTED } — **Sense of motor movement**

This value allows to invert the definition of positive direction for motor movements. Note that the limit switch signal Lim+ always blocks motion in the positive direction while Lim- blocks negative movements.

MREGMODE { EXT | CURR | TORQUE} — **Motor regulation mode**

This value selects the type of power regulation in the motor. In the current firmware version only current regulation (CURR) is implemented. If this parameter is set to EXT, the board disables its internal power driver and assumes that the motor power is applied by an external driver module.

NVOLT <float>	**Nominal operation voltage (volts)**
IVOLT <float>	**Idle operation voltage (volts)**
NCURR <float>	**Nominal current (amps)**
ICURR <integer>	**Idle current (%)**
BCURR <integer>	**Boost current increment (%)**

This parameters set the motor voltage and current values. During movements, the driving voltage and phase current are set to NVOLT (in volts) and NCURR (in amps) respectively. When the motor is stopped the voltage and current are set to IVOLT (in volts) and ICURR. Note that ICURR is not specified in amps, but in a given percentage of the nominal current NCURR.
It is possible to increase the phase current during acceleration and decceleration phases by specifying a boost current increment BCURR greater than zero. BCURR is also specified in percentage of NCURR and adds to the nominal current.

CURRGAIN { CUSTOM | LOW | MEDIUM | HIGH }	**Current regulation gain**
MREGP <float>	**Proportional coefficient**
MREGI <float>	**Integral coefficient**
MREGD <float>	**Derivative coefficient**

MREGP, MREGI and MREGD are the PID coefficients used for motor current regulation. If CURRGAIN is set to CUSTOM, the PID values can be freely set. If CURRGAIN is set to LOW, MEDIUM or HIGH, the PID values are forced to predefined values.

NRES <float>	**Nominal phase resistance**

This parameter sets the value of the nominal electrical resistance of the motor phases in ohms.
If NRES is set to zero, ...

INDEXER { INTERNAL | InPos | EncIn }	**Default indexer source**

Selects if the axis must be operated by using the internal built-in indexer for trajectory generation or an external signal applied to the InPos or EncIn inputs. This parameter refers to the default value, the actual indexer source can be changed during operation.

SHFTENC { NONE | InPos | EncIn | AbsEnc}	**Shaft encoder**
TGTENC { NONE | InPos | EncIn | AbsEnc}	**Target encoder**
CTRLENC { NONE | InPos | EncIn | AbsEnc}	**Control encoder**

Select which input position signals will be used as shaft encoder, target encoder and control encoder. If any of these parameters is set to NONE the corresponding function is left unassigned.

POSSRC { INDEXER | SHFTENC | TGTENC }	**Nominal axis position source**

Selects which functional signal, indexer, shaft encoder or target encoder, will be used to determine the nominal position of the axis.

ANTURN <integer>	**Axis reference number of turns**
ANSTEP <integer>	**Axis reference number of units/steps**

Defines the resolution of the axis by specifying the number of units/steps (ANSTEP) for a given number of motor turns (ANTURN). This resolution can be selected independently of the actual resolution of the signal selected as position source by the POSSRC parameter. If both resolutions are different, the position values are internally converted whenever they are queried or set by the appropriate commands.
Note however that the resolution of the internal indexer is always set to the value defined by the ANTURN and ANSTEP parameters.

DEFVEL <float>	**Default velocity (steps/sec)**
DEFACCT <float>	**Default acceleration time (sec)**

Configures the default values for velocity and acceleration time. The velocity value are specified in axis units (or steps) per second. The acceleration time is specified in seconds.

STRTVEL <float>	**Maximum start velocity (steps/sec)**

Configures the maximum starting velocity. This value, that is specified in axis units (or steps) per second, is the maximum velocity that can be applied to the motor without acceleration ramp. It is only used when the driver has to limit the motor slew rate as it is required in some closed loop modes for instance.

CTRLERROR <integer>	**Maximum control encoder error (steps)**

Configures the ...

PCLOOP { OFF | TGTENC }	**Default position closed loop mode**
PCLTAU <float>	**Position closed loop time constant (sec)**
PCLERROR <integer>	**Maximum closed loop error (steps)**
PCLDEADBD <integer>	**Minimum closed loop error (steps)**
PCLSETLW <integer>	**Closed loop settling window (steps)**
PCLSETLT <float>	**Closed loop settling time (sec)**

The position closed loop default mode and encoder signal is selected by the PCLOOP parameter. The position closed loop is an integral correction algorithm that forces the target encoder signal to follow the desired axis value with the regulation time constant set by PCLTAU. The maximum acceptable follow error is the number of axis steps in PCLERROR. If the error is less than PCLDEADBD the correction algorithm does not take any action. If the error is less than PCLSETLW, the driver status bit INWINDOW will be set. At the end of a movement the driver status bit SETTLING will be set until the error remains less than PCLSETLW during the time defined by PCLSETLT.

LPPOL { NORMAL | INVERTED }	**Polarity of the Lim+ signal**
LMPOL { NORMAL | INVERTED }	**Polarity of the Lim- signal**
HOMEPOL { NORMAL | INVERTED }	**Polarity of the Home signal**

These parameters allow to invert the electrical polarity (logic value) of the limit switch signals and the Home input. Note that LPPOL and LMPOL do not change the functional assignment of the limit switches: Lim+ always blocks motion in the positive direction while Lim- blocks always negative movements.

EINTURN <integer>	**EncIn reference number of turns**
EINSTEP <integer>	**EncIn reference number of units/steps**
INPNTURN <integer>	**InPos reference number of turns**
INPNSTEP <integer>	**InPos reference number of units/steps**
ABSNTURN <integer>	**AbsEnc reference number of turns**
ABSNSTEP <integer>	**AbsEnc reference number of units/steps**

Allow to define the resolution of the encoders connected to the physical encoder inputs: EncIn, InPos and AbsEnc. The resolution is defined by specifying the number of encoder units/steps (*encoder*NSTEP) for a given number of motor turns (*encoder*NTURN). These values must match the resolution of the encoders in the actual mechanics.

EINMODE { QUAD | PULSE+ | PULSE- }	**EncIn input counting mode**
INPMODE { QUAD | PULSE+ | PULSE- }	**InPos input counting mode**

Select the input counting mode (quadrature counting or pulse/direction) for the incremental encoder signals connected to the position inputs EncIn and InPos. In the case of pulse/direction counting mode, it is possible to select if the incremental counting takes place at the rise edge (PULSE+) or the falling edge (PULSE-) of the pulse signal.

EINSENSE { NORMAL | INVERTED }	**EncIn sense**
INPSENSE { NORMAL | INVERTED }	**InPos sense**

Allow to change the sign of the incremental encoder signals EncIn and InPos. Inverting the sign of the incremental signal is equivalent to invert the sense of motion of the encoder.

EINAUXPOL { NORMAL | INVERTED }	**Polarity of the EncInAux signal**
INPAUXPOL { NORMAL | INVERTED }	**Polarity of the InPosAux signal**

These parameters allow to invert the electrical polarity (logic value) of the auxiliary signals EncInAux and InPosAux.

ABSSENSE { NORMAL | INVERTED }	**AbsEnc sense**
ABSOFSSET <integer>	**AbsEnc position offset**

Allow to apply a sign inversion and an offset to the absolute encoder value read through the SSI encoder interface.

SSIDBITS <integer>	**SSI data bits**
SSICODE { BINARY | GRAY }	**SSI data coding**
SSISTATUS { S | .S | ES | OS }	**SSI status/control bits**
SSICLOCK { 125KHz | 250KHz | 500KHz | 1.25MHz | 2.5MHz | 5MHz | 12.5MHz | 25MHz | OFF}	**SSI clock frequency**
SSIDELAY { 0 | 5us | 10us | 20us | 30us | 50us | 100us | 500us}	**SSI polling delay**

Configuration parameter for the SSI interface.

HOMESRC { Lim+ | Lim- | Home | EncAux | InpAux}	**Homing signal**
HOMETYPE { LEVEL | PULSE | MPULSE }	**Type of homing signal**

The parameter HOMESRC selects the hardware signal to be used by the homing procedure. The type of signal, i.e how the homing signal indicates the reference mechanical position, is configured by HOMETYPE. Possible values are LEVEL, if the signal logic level changes at the reference position, PULSE if the signal is a short pulse at the reference position, or MPULSE if the homing device produces multiple pulses with variable distances between them. See for details

HOMEFLAGS [AUTODIR] [REVERSE] [SETPOS] [SLOW] [NEGEDGE]	**Homing flags**

Configuration of the behaviour of the homing functionality.

HOMEPOS <integer>	**Reference homing position**

Configuration of ...

HOMEVEL <float>	**Slow homing velocity (steps/sec)**

Configuration of ...

OUTPSRC { AXIS | INDEXER | SHFTENC | TGTENC | InPos | EncIn | Sync}	**OutPos source signal**
OUTPMODE { QUAD | PULSE+ | PULSE- }	**OutPos output counting mode**
OUTPPULSE { 50ns | 200ns | 2us | 20us}	**OutPos pulse width**
OUTPSENSE { NORMAL | INVERTED }	**OutPos sense**

Configuration of the OutPos position output signal.

OUTPAUXSRC {LOW | HIGH | Lim+ | Lim- | Home | EncAux | InpAux | SyncAux"}	**OutPosAux source signal**
OUTPAUXPOL { NORMAL | INVERTED }	**Polarity of the OutPosAux signal**

Configuration of the OutPosAux auxiliary output signal.

INFASRC { LOW | HIGH | Lim+ | Lim- | Home | EncAux | InpAux | SyncAux | ENABLE | ALARM | READY | MOVING | BOOST | STEADY}	**InfoA source signal**
INFBSRC { LOW | HIGH | Lim+ | Lim- | Home | EncAux | InpAux | SyncAux | ENABLE | ALARM | READY | MOVING | BOOST | STEADY}	**InfoB source signal**
INFCSRC { LOW | HIGH | Lim+ | Lim- | Home | EncAux | InpAux | SyncAux | ENABLE | ALARM | READY | MOVING | BOOST | STEADY}	**InfoC source signal**
INFAPOL { NORMAL | INVERTED }	**InfoA polarity**
INFBPOL { NORMAL | INVERTED }	**InfoB polarity**
INFCPOL { NORMAL | INVERTED }	**InfoC polarity**

Configuration of InfoA, InfoB and InfoC output signals.

EXTALARM { NONE | ...}	**External alarm signal**
EXTWARNING { NONE | ...}	**External warning signal**
Selection of the external alarm and warning signals.	

4. COMMUNICATION PROTOCOL

4.1. Communication basics

This section covers the IcePAP communication protocol. The communication interface is implemented at the system master board.

Communication is achieved by bi-directional byte streams. Normal command and response messages are transferred as lines of printable ASCII characters. The only exception is the transfer of binary data blocks, a special feature described in 4.5.
Commands messages sent to IcePAP must be formatted as sequences of printable characters terminated by a "*carriage return*" (ASCII 0x0D). Any additional control character, like "*line feed*", is ignored.
Response messages produced by the device consist on lines terminated by a "*carriage return*" + "*line feed*" character sequence (ASCII 0x0D 0x0A).

4.1.1. System commands

Commands that ... These commands are processed by the system master.

4.1.2. Board commands

Commands addressed to specific boards. Both controller and drivers

4.1.3. Local driver interface

Each driver board has an individual communication port for diagnostic purposes. It can also be used for stand alone operation (no further discussed).

4.2. Interfaces

The master boards integrate three communication ports: an Ethernet interface, a serial line and an USB port. The characteristics of the different interfaces are the following:

Interface	*Type*	*Parameters*
Serial Line	**RS232**	9600bauds, No parity, 1 stop bit
Ethernet	**100baseT**	TCP sockets, port 5000
Universal Serial Bus	**USB 1.0**	Not implemented in the current version

4.3. Syntax conventions

In the most usual case remote control is implemented by an application program running in a host computer that sends commands and requests to the *isgdevice* as sequences of ASCII characters. The syntax rules are described below. See X for practical examples.

4.3.1. Commands and requests

- Command lines consist of a command keyword optionally followed by parameters.
  - The number and type of parameters depend on the particular command.
- Command keywords are not case sensitive.
  - The device converts internally all the characters to uppercase before any syntax checking. (TO BE DISCUSSED)

- Parameters are also converted to uppercase unless they are enclosed between double quotes (″″, ASCII 0x22). (TO BE DISCUSSED)

- Commands may be optionally preceded by the acknowledge character.
  - The acknowledge character is a hash symbol (#, ASCII 0x23) that must appear in the command line immediately before the first character of the command keyword.

- Normal (non query) commands never produce response messages unless the acknowledge character is used.
  - Non query command keywords always start by an alphabetical character (A to Z). Exceptions are binary transfer commands (see XX) that start by an asterisk character (*, ASCII 0x2A).
  - If the acknowledge character is used, the device produces the response string OK if the command execution was successful.
  - If the acknowledge character is used and the command does not executes successfully, the device produces either the string ERROR or a string containing a human readable error message. The behaviour depends on the current setting of the *echo* mode (see 4.4).

- Requests are query commands that produce response messages from the device.
  - Requests keywords always start by a question mark character (?, ASCII 0x3F).
  - If the request is successful the content of the response message depends on the particular request.
  - If request fails the device produces either the string ERROR or a string containing a human readable error message. The behaviour depends on the current setting of the *echo* mode (see 4.4).
  - The acknowledge character has no effect when used with requests.

- Response messages consist of one or more ASCII character lines.
  - The way every line in a response message is terminated depends on the type of communication port (see **Error! Reference source not found.**).
  - A response message may contain either the output of a request, an acknowledgement keyword (OK or ERROR) or a human readable error message.
  - When a response message consists of more than one line, the first and last lines contain a single dollar character ($, ASCII 0x3F).

4.3.2. Addressing

- Board commands must be sent to the specific controller or drivers boards by using an addressing prefix. An addressing prefix consists of the board address in decimal format followed by a colon character (:, ASCII 0x3A). No spaces are allowed between the last address digit and the colon character.

- An addressing prefix consisting of only the colon character (:) with no address string is interpreted as a broadcast command. In that case the command is forwarded to all the boards in the system. Controller boards ignore broadcasts of driver-only commands as well as driver boards ignore controller-only broadcasts. No queries or acknowledge characters are allowed in broadcasts.

4.4. Terminal mode

When an IcePAP system is accessed through a serial port, two possible communication modes are available that can be selected with the commands `ECHO` and `NOECHO`. The differences between these two modes are described below. This commands can be issued through other interfaces (i.e. Ethernet) but they only have effect on the serial port.

Echo mode (terminal mode)

This mode should be used when the IcePAP master board is connected to a dumb terminal. In this case the user types commands on the keyboard and reads the answers and error messages on the terminal screen without computer intervention. This mode is usually not active by default and the user has to send the `ECHO` command every time the device is powered on.

In echo mode all the characters sent to the device are echoed back to the terminal. The device also sends human-readable messages to be printed on the terminal screen whenever an error is detected in commands or requests.
Case conversion takes place before the characters are sent back to the terminal, therefore characters are echoed back as uppercase even if they are typed and sent to the device as lowercase. (TO BE DISCUSSED)
In echo mode the backspace character (ASCII 0x08) has the effect of deleting the last character received by the device. In this way a minimum editing functionality is provided.

Noecho mode (host computer)

This is the default mode. In this case no characters are echoed and no error messages are returned by non-query commands unless they are explicitly requested by the acknowledge character. This mode is intended to be used when a program running in a host computer communicates with the controller, sending commands and analysing the answers.

4.5. Binary transfer

Binary transfer is a special mode that extends the standard protocol allowing faster data transfer. Binary blocks have a maximum size of 65535 data bytes (0xFFFF).

Binary transfer commands or requests are initiated by ASCII command lines that follow the same rules than ordinary commands or requests (see 4.3.1). The only difference is that binary transfer command lines must include an asterisk character (*, ASCII 0x2A) in the command or request keyword. Non-query commands keywords must start by an asterisk character. Request keywords must include the asterisk as the first character after the question mark.

Once the *isgdevice* has received the ASCII command line, the data is transferred as a binary block. In the case of non-query commands, the binary data block is sent from the host computer to the device. In case of binary requests, the device sends the binary block to the host (serial line) or puts it in its output buffer ready to be read by the host (GPIB).

If the device finds an error in a command line containing a binary request, instead of the binary block, it produces the string `ERROR`.
The acknowledge character (`#`, ASCII 0x23) can be used in the same way that with non-binary commands. If it is included in a non-query command line, the device produces an acknowledgement keyword (`ERROR` or `OK`) to signal if the command line contained errors or not. The acknowledge character has no effect in the case of binary requests.

Although binary transfer is initiated in the same way for both serial line and GPIB communication, the format of the binary data blocks and the management of the end of transfer condition are different in both cases.

4.5.1. Serial port binary blocks

In the case of transfer through a serial port, the binary block contains the binary data and 4 extra bytes. The structure of the block is the following:

byte Number	content
0	0xFF (signature)
1	DataSize (MSB)
2	DataSize (LSB)
3	data byte (first)
…	…
DataSize + 2	data byte (last)
DataSize + 3	Checksum

The first byte contains always the value 0xFF (255) and can be used the signature of the block. The next two bytes contain the number of data bytes to transfer. The last byte contains the check sum value that is used to verify data integrity.

The checksum value is calculated as the lower 8-bits of the sum of all the bytes in the binary block with exception of the signature byte (and the checksum byte itself).

4.5.2. TCP binary blocks

In the case of transfer by Ethernet, the binary block does not contain any additional control or protocol byte. Only the actual data bytes are transferred. The EOI line is asserted during the transfer of the last data byte to signal the end of the transmission.

5. COMMAND SET

BOARD COMMANDS

Command		Description	Controller	Driver	Page
	?ACTIVE	Query activation status		❒	31
	?MODE	Query board mode	❒	❒	64
	?STATUS	Query board status	❒	❒	84
	?VSTATUS	Query verbose board status	❒	❒	90
	?ALARM	Query board alarm message	❒	❒	34
	?WARNING	Query board warnings	❒	❒	91
WTEMP	?WTEMP	Set/query warning temperature	❒	❒	92
CONFIG	?CONFIG	Manage configuration mode		❒	42
CFG	?CFG	Set/query configuration parameters		❒	37
	?CFGINFO	Query configuration parameter info		❒	39
	?VER	Query board version information	❒	❒	89
NAME	?NAME	Set/query board name		❒	66
	?ID	Query board identification	❒	❒	57
	?POST	Query power-on selftest results	❒	❒	71
POWER	?POWER	Set/query motor power state		❒	72
AUXPS	?AUXPS	Set/query auxiliary power supply state		❒	35
	?MEAS	Query measured value	❒	❒	62
POS	?POS	Set/query axis position in axis units	❒	❒	69
ENC	?ENC	Set/query axis position in encoder steps	❒	❒	45
	?HOMESTAT	Query home search status		❒	56
	?HOMEPOS	Query the found home position in axis units		❒	55
	?HOMEENC	Query the found home position in encoder steps		❒	54
VELOCITY	?VELOCITY	Set/query programmed axis velocity	❒	❒	88
ACCTIME	?ACCTIME	Set/query acceleration time	❒	❒	32
PCLOOP	?PCLOOP	Set/query current position closed loop mode		❒	68
ESYNC		Synchronise internal position registers		❒	48
MOVE		Start absolute movement	❒	❒	65
RMOVE		Start relative movement	❒	❒	78
JOG	?JOG	Set/query jog velocity	❒	❒	61
HOME		Start home signal search sequence		❒	53
CMOVE		Start relative movement in configuration mode		❒	41
CJOG		Set jog velocity in configuration mode		❒	40
STOP		Stop movement	❒	❒	85
ABORT		Abort movement	❒	❒	30
INDEXER	?INDEXER	Set/query indexer signal source		❒	58
INFOA	?INFOA	Set/query InfoA signal source and polarity		❒	59
INFOB	?INFOB	Set/query InfoB signal source and polarity		❒	59
INFOC	?INFOC	Set/query InfoC signal source and polarity		❒	59
	?HELP	Query list of available commands	❒	❒	52
	?ERRMSG	Query last command error message	❒	❒	47
	?FERRMSG	Query first error message	❒	❒	49
BLINK	?BLINK	Set/query remaining blinking time	❒	❒	36
	?TIME	Query running time	❒	❒	87
DEBUG	?DEBUG	Set/query debug level	❒	❒	43
ECHO		Select echo mode		❒	44
NOECHO		Cancel echo mode		❒	67
	?MEMORY	Query available memory	❒	❒	63
	?ADDR	Query board address	❒	❒	33

SYSTEM COMMANDS

Command		Description	Page
MODE	?MODE	Set/query system mode	64
	?SYSSTAT	Query system configuration	86
	?STATUS	Query multiple board status	84
	?FSTATUS	Query multiple board fast status	51
REPORT	?REPORT	Set/query asynchronous report settings	74
	?VER	Query system firmware version information	89
	?RID	Query rack identification string	76
	?RTEMP	Query rack temperatures	79
*PROG PROG	?PROG	Firmware programming	73
RFPROG		Factory firmware programming	77
RESET		System or rack reset	75
POS	?POS	Set/query multiple axis position in axis units	69
ENC	?ENC	Set/query multiple axis position in encoder steps	45
	?FPOS	Fast query of multiple board positions	50
	?HOMESTAT	Query multiple axis home search status	56
	?HOMEPOS	Query the found multiple axis home position in axis units	55
	?HOMEENC	Query the found multiple axis home position in encoder steps	54
VELOCITY	?VELOCITY	Set/query programmed multiple axis velocity	88
ACCTIME	?ACCTIME	Set/query acceleration time	32
MOVE		Start multiple axis absolute movement	65
RMOVE		Start multiple axis relative movement	78
JOG	?JOG	Set/query multiple axis jog velocities	61
HOME		Start multiple axis home signal search sequence	53
STOP		Stop multiple axis movement	85
ABORT		Abort movement	30
	?HELP	Query list of available commands	52
	?ERRMSG	Query last command error message	47
ECHO		Select serial line echo	44
NOECHO		Cancel serial line echo	67

5.1. Command reference

ABORT

Abort movement

Syntax:

<board_addr>:ABORT **(board command)**

or

ABORT [<axis1> [<axis2> … [<axisN>]…]] **(system command)**

Description:

The ABORT command aborts all movement in the specified axis.

When using the system command, if the command can not be issued for a certain axis, all the movements in the system will be aborted.

If one of the explicitly aborted axis belongs to a predefined group, all the members of that group will be aborted.

Examples:

```
Command:  16:ABORT
Command:  ABORT                    // abort all the axes in the system
Command:  ABORT 30 33 42           // abort axes 30, 33 and 42

Command:  #ABORT 30 33 42
Answer:   ABORT ERROR All axes aborted. Axis 33: Board is not
          present in the system

Command:  #ABORT 30 rrt 42
Answer:   ABORT ERROR All axes aborted. Wrong parameter(s)
```

?ACTIVE

Query activation status

Syntax:

<driver_addr>:?ACTIVE

Answer:

<driver_addr>:?ACTIVE { YES | NO }

Description:

Returns the current activation status of a driver board. A driver will be active if the internal ACTIVE configuration parameter is set to YES and the board is not in PROG or TEST mode. Otherwise the ?ACTIVE query will return NO.

When a driver board is not active, the motor power and the trajectory generation functions are disabled. (to be checked)

The driver's ACTIVE configuration parameter will be reset to the value NO at power on if:

- the address of the driver has changed, for instance if the board has been moved to a different slot within the same system,
- the driver board finds itself plugged in a different IcePAP system, i.e. the master crate AND the master controller are different than the previous ones
- there is a firmware or a command set mismatch between the driver and the master controller board.

A standalone driver (with no master controller) can be activated through the front panel serial line with the command SLACT.

Examples:

Command: `16:?ACTIVE`

Answer: `16:?ACTIVE YES`

ACCTIME / ?ACCTIME

Set/query acceleration time

Syntax:

<board_addr>:ACCTIME [<accTime>] **(board command)**

or

ACCTIME [<axis1> <accTime1> … [<axisN> <accTimeN>]…] **(system command)**

Description:

Sets the acceleration time for the corresponding axis to the <accTime> values in seconds. The actual acceleration for each axis is calculated internally based on the current value of the axis velocity (see VELOCITY command).

The acceleration time is internally recalculated every time that the axis velocity changes.

Syntax:

<board_addr>:?ACCTIME **(board command)**

or

?ACCTIME [<axis1> [<axis2> … [<axisN>]…]] **(system command)**

Answer:

<board_addr>:?ACCTIME <accTime> **(board answer)**

or

?ACCTIME <accTime1> <accTime2> … <accTimeN> **(system answer)**

Description:

Returns the current acceleration time of the specified axes in seconds.

Examples:

Command: `16:?ACCTIME`

Answer: `16:?ACCTIME 0.25`

Command: `24:ACCTIME 0.1`

Command: `?ACCTIME 16 24`

Answer: `?ACCTIME 0.25 0.1`

Command: `ACCTIME 16 0.1 17 0.2`

?ADDR

Query board address

Syntax:

<board_addr>:?ADDR

Answer:

<board_addr>:?ADDR <boardAddr>

Description:

The ?ADDR command returns the current board address. This command is only useful when the board is accessed through the local serial line interface.

Examples:

Command: `16:?ADDR`

Answer: `?ADDR 16` // useless information

Access through the local serial line interface:

Command: `?ADDR`

Answer: `?ADDR 16`

?ALARM

Query board alarm message

Syntax:

<board_addr>:?ALARM

Answer:

<board_addr>:?ALARM { NO | <alarm_condition string> }

Description:

If the board is disabled by an alarm condition, this query returns a string describing the such a condition. If not, the query returns the string NO.

Possible alarm conditions are:

Alarm Condition	*Description*
Internal failure	Power-on self test (POST) or motor supply failure
Motor failure	Error detectred in motor connection
Power overload	Current of power overload
Driver overheating	The temperature of the driver board exceeds maximum value
Close loop error	The closed loop follow error exceeds the maximum value (see PCLERROR configuration parameter)
Control encoder error	The control encoder discrepancy exceeds the CTRLERROR configuration parameter
External alarm	The external alarm signal is active (see EXTALARM configuration parameter)

Examples:

```
Command:    115:?ALARM
Answer:     115:?ALARM <alarm condition string>
Command:    115:#POWER ON
Answer:     115:POWER OK
Command:    115:?ALARM
Answer:     115:?ALARM NO
```

AUXPS / ?AUXPS

Set/query axis auxiliary power supply state

Syntax:

<driver_addr>:AUXPS [{ON | OFF}]

Description:

Switches on or off the auxiliary power supply in a driver board. When the auxiliary power supply is switched off, the motor power is also switched off.

Syntax:

<driver_addr>:?AUXPS

Answer:

<driver_addr>:?AUXPS [{ON | OFF}]

Description:

Returns the state of the auxiliary power supply of the driver board.

Examples:

Command:	`83:?AUXPS`
Answer:	`83:?AUXPS ON`
Command:	`83:AUXPS OFF`
Command:	`83:?AUXPS`
Answer:	`83:?AUXPS OFF`

BLINK / ?BLINK

Set/query remaining blinking time

Syntax:

<board_addr>:BLINK <blinkTime>

Description:

If <blinkTime> is greater than zero, sets the board in blinking mode for a period given by <blinkTime> in seconds. If <blinkTime> is zero, this command stops blinking mode.

Syntax:

<board_addr>:?BLINK

Answer:

<board_addr>:?BLINK <remBlinkTime>

Description:

Returns the remaining blinking time.

Examples:

Command:	`83:BLINK 10`
Command:	`83:?BLINK`
Answer:	`83:?BLINK 8.4532`
Command:	`83:?BLINK`
Answer:	`83:?BLINK 6.5439`

CFG / ?CFG

Set/query configuration parameters

Syntax:

<driver_addr>:CFG <configPar> <configVal>

or

<driver_addr>:CFG { DEFAULT | EXPERT }

Description:

The CFG command allows to change the current values of the configuration parameters of a driver board. The driver has to be previously switched into configuration mode (see CONFIG command).

The configuration of driver boards as well as the list of available parameters is detailed in chapter 2.

The command CFG DEFAULT instructs the driver board to revert all its configuration parameters to the default values. The list of default values can be obtain from the driver by means of the ?CFG DEFAULT query.

The command CFG EXPERT sets an internal flag that can be read back with the ?CFG query. This flag has not any specific function in the IcePAP system but it is provided as an facility to external configuration tools to confirm the validity of the current driver configuration when the driver boards are moved among systems. As the expert flag is cleared by any other CFG command, it must be set immediately before the configuration is validated by the CONFIG command.

Syntax:

<driver_addr>:?CFG [<configPar> | DEFAULT | EXPERT]

Answer:

<driver_addr>:?CFG <configPar> <configVal>

or

<driver_addr>:?CFG $

<configPar1> <configVal1>

<configPar2> <configVal2>

…

<configParN> <configValN>

$

Description:

The ?CFG query returns the value <configVal> assigned to a particular configuration parameter <configPar>. If no parameter is specified, the query returns a multiline answer with the complete list of configuration parameters and their current values. If the DEFAULT keyword is used, instead of the current values, the ?CFG query returns the complete list of configuration parameters and their default values.

If the EXPERT keyword is used as a parameter name, the ?CFG query returns the value of the internal expert flag set by the CFG EXPERT command and cleared by any other CFG command. The value is returned as a YES/NO boolean value. Note however that EXPERT is not a configuration parameter.

Examples:

Command: `15:CFG DEFAULT`

Command: `15:?CFG NCURR`

Answer: `15:?CFG NCURR 0.1`

Command: `15:CFG NCURR 2.4`

Command: `15:?CFG`

Answer:
```
15:?CFG $
ACTIVE NO
PROTLEVEL 0
NAMELOCK NO
POWERON NO
MOTPHASES 2
MOTORSENSE NORMAL
MOTPOLES 50
MREGMODE CURR
...
NCURR 2.4
...
INFCSOURCE Home
INFCPOL NORMAL
$
```

Command: `23:?CFG EXPERT`

Answer: `23:?CFG EXPERT NO`

?CFGINFO

Query configuration parameter info

Syntax:

<driver_addr>:?CFGINFO [<configPar>]

Answer:

<driver_addr>:?CFGINFO <configPar> {INTEGER | FLOAT | *labelList* }

or

<driver_addr>:?CFGINFO $

<configPar1> {INTEGER | FLOAT | *labelList1* }

<configPar2> {INTEGER | FLOAT | *labelList2* }

…

<configParN> {INTEGER | FLOAT | *labelListN* }

$

Where *labelList* is a list of character strings separated by whitespaces and enclosed in curly braces ({}). [FLAG LIST descriptionmissing]

Description:

The ?CFGINFO query returns the type of the configuration parameter <configPar>. Possible types are numeric (*INTEGER* or *FLOAT*) or string. In the case of strings the query returns the list of acceptable values.

If no parameter is specified, the query returns a multiline answer with the complete list of type information for all the driver configuration parameters.

Examples:

Command: `7:?CFGINFO NCURR`

Answer: `7:?CFGINFO FLOAT`

Command: `103:?CFGINFO`

Answer:
```
103:?CFGINFO $
ACTIVE {NO YES}
PROTLEVEL INTEGER
NAMELOCK {NO YES}
POWERON {NO YES}
MOTPHASES {1 2 3}
MOTORSENSE {NORMAL INVERTED}
...
INFCPOL {NORMAL INVERTED}
$
```

CJOG

Set jog velocity in configuration mode

Syntax:

<board_addr>:CJOG <signedVelocity>

Description:

Sets the specified axis or axes in jog mode at the given velocity in steps per second. This command can only be executed when the driver is in configuration mode.

The sign of the velocity parameter selects the actual direction of the movement. If a specified axis is already jogging at a certain speed, the speed will be ramped up or down as needed to reach the new velocity value. The acceleration is fixed as the ratio of the current values of axis velocity and acceleration time (see ?VELOCITY and ?ACCTIME queries). A zero velocity value forces an axis to stop.

Examples:

Command:	`5:CJOG 100`
Command:	`5:?JOG`
Answer:	`5:?JOG 100`
Command:	`#5:CJOG 200`
Answer:	`5:CJOG OK`
Command:	`#5:CJOG -200`
Answer:	`3:CJOG ERROR Cannot change jog direction`

CMOVE

Start absolute movement in configuration mode

Syntax:

<driver_addr>:CMOVE <absolutePos>

Description:

Performs an absolute movement on the specified driver board. This command can only be executed when the driver is in configuration mode.

Examples:

Command: `115:CMOVE -7000`

Command: `115:?POS`

Answer: `115:?POS -7000`

CONFIG / ?CONFIG

Manage configuration mode

Syntax:

<driver_addr>:CONFIG [<confID>]

Description:

The CONFIG command allows to switch a driver board into configuration mode. A driver board cannot be switched into configuration mode when the IcePAP system is in *PROG* or *TEST* modes (see MODE command).

When a driver is in configuration mode, the driver configuration parameters can be modified with the CFG command. Once the configuration has been modified, the CONFIG command, issued with a non empty <confID> string as parameter, validates the current configuration and stores it in the internal non volatile memory of the driver board. The <confID> string is also stored in the driver and can be used to identify the particular set of configuration parameters. The board also switches back to *OPER* mode.

If the driver is in configuration mode and the CONFIG command is issued with no parameters, the driver goes back to OPER mode and the last valid configuration before entering CONFIG mode is reloaded. In that case the most recent changes done during configuration mode are lost.

Syntax:

<driver_addr>:?CONFIG

Answer:

<driver_addr>:?CONFIG <confID>

Description:

The ?CONFIG query returns the identifier of the last valid configuration parameter set.

Examples:

```
Command:  32:?CFG ACTIVE
Answer:   32:?CFG ACTIVE NO
Command:  ?MODE
Answer:   ?MODE OPER                 // System mode is OPER
Command:  32:CONFIG                  // Switch axis 32 into CONFIG mode
Command:  32:?MODE
Answer:   32:?MODE CONFIG
Command:  32:CFG ACTIVE YES          // Change configuration parameter
Command:  32:CONFIG CONF001          // Validate driver configuration
Command:  32:?CFG ACTIVE
Answer:   32:?CFG YES
```

DEBUG / ?DEBUG

Set/query debug level

Syntax:

<board_addr>:DEBUG <debugLevel>

Description:

Sets the level of the debug facility to <debugLevel>. If the level is set to 0, the debug facility is switched off.

The debug level is stored in the board non-volatile memory and it is maintained after board reset.

Syntax:

<board_addr>:?DEBUG

Answer:

<board_addr>:?DEBUG <debugLevel>

Description:

Returns the current level of the debug facility.

Examples:

Command: `15:?DEBUG`

Answer: `15:?DEBUG 0`

Command: `15:DEBUG 2`

Command: `15:?DEBUG`

Answer: `15:?DEBUG 2`

ECHO

Set echo mode

Syntax:

ECHO **(system command)**

or

<board_addr>:ECHO **(board command)**

Description:

Switches the echo mode on. Useful when accessing boards through the serial line.

Example:

Command: `ECHO`

Command: `92:ECHO`

ENC / ?ENC

Set/query axis position in encoder steps

Syntax:

<board_addr>:ENC [*pos_sel*] <posVal> **(board command)**

or

ENC [*pos_sel*] <axis1> <posVal1> … <axisN> <posValN> **(system command)**

Description:

Loads the position registers in the specified boards with the <posVal> values. The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	Points to the register configured as POSSRC
INDEXER	Points to the register used as indexer (see INDEXER command)
EXTERR	[???]
SHFTENC *	Points to the register configured as SHFTENC
TGTENC *	Points to the register configured as TGTENC
ENCIN *	ENCIN register
INPOS *	INPOS register
ABSENC *	ABSENC register
MOTOR *	MOTOR register

* Only valid for driver boards

If position is not specified, the value is loaded in the axis position register

---

Syntax:

<board_addr>:?ENC [*pos_sel*] **(board query)**

or

?ENC [*pos_sel*] <axis1> <axis2> … <axisN> **(system query)**

Answer:

<board_addr>:?ENC <posVal> **(board answer)**

or

?ENC <posVal1> <posVal2> … <posValN> **(system answer)**

Description:

Returns the current signal source used as axis indexer.

Examples:

Command: `115:ENC AXIS 500`

Command: `115:ENC INDEXER -3000`

Command: `115:?ENC`

Answer: `115:?ENC 500`

Command: `?ENC INDEXER 5 115`

Answer: `?ENC 13467895 -3000`

?ERRMSG

Query last command error message

Syntax:

?ERRMSG **(system or local board query)**

Answer:

?ERRMSG [<errorMessage>] **(system or local board answer)**

Description:

If the previous command produced an error, the ?ERRMSG query returns the error message as an ASCII string. If the previous command was successful, the ?ERRMSG query returns and empty string.

This command will retrieve the last error, regardless whether the previous command was a system command or a board command.

The system does not accept it as board command if it is not issued through the serial line.

Example:

```
Command:    ?VER
Answer:     ?VER 1.00
Command:    ?ERRMSG
Answer:     ?ERRMSG
Command:    15:VELOCITY 0
Command:    15:?ERRMSG
Answer:     15:?ERRMSG Out of range value
```

ESYNC

Synchronise internal position registers

Syntax:

<driver_addr>:ESYNC

Description:

This command forces the value in all the position register that are linked to the axis to be synchronised with the axis position source.

[TODO: needs further explanation]

Examples:

```
Command:    11:?POS INDEXER
Answer:     11:?POS 1364
Command:    11:?POS TGTENC
Answer:     11:?POS 1232
Command:    11:?CFG POSSRC
Answer:     11:?CFG POSSRC INDEXER

Command:    #11:ESYNC
Answer:     11:ESYNC OK

Command:    11:?POS INDEXER
Answer:     11:?POS 1364
Command:    11:?POS TGTENC
Answer:     11:?POS 1364
```

?FERRMSG

Query first error message

Syntax:

<board_addr>:?FERRMSG

Answer:

<board_addr>:?FERRMSG [command <errorMessage>]

Description:

Returns the message for the first command error that was produced since the last time the ?FERRMSG query was issued. The query returns the command that produced the error and the error message an as ASCII string.

[TODO: Explain difference: system errors, board errors].

Example:

Command:	`?VER`
Answer:	`?VER 1.00`
Command:	`?FERRMSG`
Answer:	`?FERRMSG`
Command:	`15:VELOCITY 0`
Command:	`15:?FERRMSG`
Answer:	`15:?FERRMSG Out of range value`

?FPOS

Fast query of multiple board positions

Syntax:

?FPOS [*pos_sel*] <axis1> <axis2> … <axisN>

Answer:

?FPOS <posVal1> <posVal2> … <posValN>

Description:

Returns the positions for the specified axes. The specific position is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	Points to the register configured as POSSRC
INDEXER	Points to the register used as indexer (see INDEXER command)

If *pos_sel* is not specified, the values returned are the axis positions

---

Examples:

Command: `?FPOS INDEXER 17 18 19`

Answer: `?FPOS 13467895 0 -3000`

Command: `?FPOS 25`

Answer: `?FPOS 5366703`

?FSTATUS

Fast query of multiple board status

Syntax:

?FSTATUS <axis1> <axis2> … <axisN>

Answer:

?FSTATUS <statusReg1> <statusReg2> … <statusRegN>

Description:

Returns the value of the current status of the selected boards as 32-bit values in C-like hexadecimal notation.

?FSTATUS is a system query that is managed exclusively by the master system controller. ?FSTATUS returns values stored in the system controller that are updated every time that any bit in the status word of a board changes.

The ?FSTATUS query is intended to be used for frequent polling from the control host, as it is faster as it presents less latency than ?STATUS, and it does not load the internal communication bus.

Example:

Command: `?FSTATUS 80 83 85`

Answer: `?FSTATUS 0x00000003 0x00000003 0x00000003`

?HELP

Query list of available commands

Syntax:

<board_addr>:?HELP **(board command)**

or

?HELP **(system command)**

Description:

Returns the list of available commands and queries. The list differs between system, controller and driver commands.

Examples:

Command: `16:?HELP`

Answer:

```
$
     RESET
            ?HDWVER
            ?STATE
            ?RETCODE
     CLEAR
            ?LIST
       RUN
$
```

HOME

Start home signal search sequence

Syntax:

<driver_addr>:HOME {+1 | 0 | -1} **(driver command)**

or

HOME [GROUP] <axis1> {+1 | 0 | -1} … <axisN> {+1 | 0 | -1} **(system command)**

Description:

Starts a home search sequence in the direction specified by the direction parameter. Positive and negative directions are selected by the values +1 and -1 respectively.

The HOME command will start a homing sequence only if the source for the homing reference signal has been previously selected by setting the HOMESRC configuration parameter to a value different from NONE. Otherwise the HOME command produces an error.

The direction parameter can be set to 0 for a given axis only if the AUTODIR flag is set in the HOMEFLAGS configuration parameter for that axis. In that case the search direction of the homing sequence is determined from the logic value of the homing reference signal as it is described in 2.2.2. If the direction parameter is set to 0 for a given axis but the AUTODIR flag is not set, the HOME command produces an error.

Examples:

Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT NOTFOUND 0`
Command:	`16:HOME +1`
Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT MOVING +1`
Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT MOVING -1`
Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT FOUND -1`

?HOMEENC

Query the found home position in encoder steps

Syntax:

<board_addr>:?HOMEENC [*pos_sel*] **(driver command)**

or

?HOMEENC [*pos_sel*] <axis1> … <axisN> **(system command)**

Description:

Query the position registers latched when the configured homing signal event arrived. The answer is a number of encoder steps.

If no homing latch event happened after the last home search, an error is issued

The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	
INDEXER	
POSERR	
SHFTENC *	
TGTENC *	
ENCIN *	
INPOS *	
ABSENC *	
MOTOR *	

* Only valid for driver boards

If pos_sel is not specified, the value returned is axis position at homing latch event.

Examples:

Command: `16:?HOMEPOS`

Answer: `16:?HOMEPOS ERROR Last home search was not successful`

Command: `16:HOME +1`

Command: `16:?HOMESTAT`

Answer: `16:?HOMESTAT FOUND -1`

Command: `16:?HOMEENC`

Answer: `16:?HOMEENC 12345`

Command: `16:?HOMEENC TGTENC`

Answer: `16:?HOMEENC 12350`

?HOMEPOS

Query the found home position in axis units

Syntax:

<board_addr>:?HOMEPOS [*pos_sel*] **(driver command)**

or

?HOMEPOS [*pos_sel*] <axis1> … <axisN> **(system command)**

Description:

Query the position registers latched when the configured homing signal event arrived. The answer is a number of steps in axis units.

If no homing latch event happened after the last home search, an error is issued

The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	
INDEXER	
POSERR	
SHFTENC *	
TGTENC *	
ENCIN *	
INPOS *	
ABSENC *	
MOTOR *	

* Only valid for driver boards

If pos_sel is not specified, the value returned is axis position at homing latch event.

Examples:

Command: `16:?HOMEPOS`

Answer: `16:?HOMEPOS ERROR Last home search was not successful`

Command: `16:HOME +1`

Command: `16:?HOMESTAT`

Answer: `16:?HOMESTAT FOUND -1`

Command: `16:?HOMEPOS`

Answer: `16:?HOMEPOS 12345`

Command: `16:?HOMEPOS TGTENC`

Answer: `16:?HOMEPOS 12350`

?HOMESTAT

Query home search status

Syntax:

<driver_addr>:?HOMESTAT **(driver command)**

or

?HOMESTAT <axis1> … <axisN> **(system command)**

Answer:

<driver _addr>:?HOMESTAT {MOVING | FOUND | NOTFOUND} {+1| 0 |-1} **(driver answer)**

or

?HOMESTAT hStatus1 hDirection1 … hStatusN hDirectionN **(system answer)**

where *home_status* is one of MOVING, FOUND or NOTFOUND

and *home_dir* is one of +1, 0 or -1

Description:

Query information about the ongoing or previous home search sequence. If the homing sequence is in progress, ?HOMESTAT returns MOVING as hStatus keyword. If the sequence is finished, the query returns either FOUND or NOTFOUND depending on whether the home search succeeded or not.

The numeric value hDirection after the hStatus keyword, represents the direction of motion, -1 or +1, either during the search sequence or when the search is successfully completed. If the homing sequence fails, the returned hDirection value is 0.

Examples:

Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT NOTFOUND 0`
Command:	`16:HOME +`
Command:	`16:?HOMESTAT`
Answer:	`16:?HOMESTAT MOVING +1`

?ID

Query board identification

Syntax:

<board_addr>:?ID [{ HW | SN }]

Answer:

<board_addr>:?ID { <hwIDstring> | <serialNumber> }

Description:

The ?ID query returns either the hardware identification string or the serial number as default the HW is return.

Examples:

Command:	`16:?ID`
Answer:	`?ID xxxx.xxxx.xxxx`
Command:	`31:?ID SN`
Answer:	`?ID 0034-44587`

INDEXER / ?INDEXER

Select/query indexer signal source

Syntax:

<driver_addr>:INDEXER [{ INTERNAL | SYNC | INPOS | ENCIN }]

Description:

Selects the signal source used for the axis indexer.

If no value is specified, the indexer source is set to the default value defined by the configuration parameters (see CFG INDEXER).

Available signal sources:

Source	***Indexer signal***
INTERNAL	Internally generated indexer is used
SYNC	*Sync* signal distributed through the rack backplane
INPOS	*InPos* signal at the front panel connector (Axis Interface)
ENCIN	*EncIn* signal at the rear panel

Syntax:

<driver_addr>:?INDEXER

Answer:

<driver_addr>:INDEXER { INTERNAL | SYNC | INPOS | ENCIN }

Description:

Returns the current signal source used as axis indexer.

Examples:

```
Command:    34:?CFG INDEXER
Answer:     34:?CFG INDEXER INTERNAL        // the default is INTERNAL
Command:    34:INDEXER SYNC                 // change the indexer source
Command:    34:?INDEXER
Answer:     34:?INDEXER SYNC
Command:    34:INDEXER                      // set the default value back
Command:    34:?INDEXER
Answer:     34:?INDEXER INTERNAL
```

INFOA / ?INFOA
INFOB / ?INFOB
INFOC / ?INFOC

Set/query info signal source and polarity

Syntax:

<driver_addr>:INFOx [*signal_source* [{NORMAL | INVERTED}]]

where INFOx is one of INFOA, INFOB or INFOC.

Description:

Configures the Info outputs. If the polarity is not specified it is set to NORMAL.
If *signal_source* is not specified, the signal is configured to the default value.
Possible values of *signal_source* are:

Source	***...***
LOW	low logic level
HIGH	high logic level
LIM+	limit- signal
LIM-	limit+ signal
HOME	home signal
ENCAUX	EncAux signal
INPAUX	InPosAux signal
SYNCAUX	SyncAux signal
ENABLE	power enable
ALARM	alarm condition
READY	axis ready
MOVING	axis moving
BOOST	axis in acceleration phase
STEADY	axis moving at constant velocity
.MAIN	internal signals (only for diagnostic)
.ISR	

Syntax:

<driver_addr>:?INFOx

where ?INFOx is one of ?INFOA, ?INFOB or ?INFOC

Answer:

<driver_addr>:INFOx *signal_source* {NORMAL | INVERTED}

where *signal_source* is one of the possible values presented above.

Description:

Returns the configuration of the corresponding Info output signal.

Examples:

Command: `12:INFOB READY`

Command: `?12:INFOB`

Answer: `?12:INFOB READY NORMAL`

JOG / ?JOG

Set/query jog velocity

Syntax:

<board_addr>:JOG <signedVelocity> **(board command)**

or

JOG [GROUP] <axis1> <signedVel1> … <axisN> <signedVelN> **(system command)**

Description:

Sets the specified axis or axes in jog mode at the given velocity in steps per second. The sign of the velocity parameter selects the actual direction of the movement. If a specified axis is already jogging at a certain speed, the speed will be ramped up or down as needed to reach the new velocity value. The acceleration is fixed as the ratio of the current values of axis velocity and acceleration time (see ?VELOCITY and ?ACCTIME queries). A zero velocity value forces an axis to stop.

If the GROUP keyword is used with the JOG system command, the set of axes is managed as a group by the system controller as described in 2.2.2.

Syntax:

<board_addr>:?JOG **(board query)**

or

?JOG <axis1> <axis2> … <axisN> **(system query)**

Answer:

<board_addr>:?JOG < signedVelocity > **(board answer)**

or

?JOG < signedVel1> < signedVel2> … < signedVelN> **(system answer)**

Description:

Returns the current jog velocities of the specified axes in steps per second. If an axis is not in jog mod, a zero velocity value is returned.

Examples:

Command:	`5:JOG 100`
Command:	`5:?JOG`
Answer:	`5:?JOG 100`
Command:	`#5:JOG 200`
Answer:	`5:JOG OK`
Command:	`#5:JOG -200`
Answer:	`3:JOG ERROR Cannot change jog direction`
Command:	`JOG 5 200 23 100`

?MEAS

Query measured value

Syntax:

<board_addr>:?MEAS { VCC | VM | I | IA | IB | IC | T | RT}

Answer:

<board_addr>:?MEAS <measuredValue>

Description:

Returns a measured value for the specified axes. The of the possible measured values and their meaning is compiled in the following table:

magnitude	*description*	*units*	*applies to:*
VCC	Main power supply voltage	volts	only drivers
VM	Motor voltage	volts	only drivers
I	Motor current	amps	only drivers
IA	Phase A current	amps	only drivers
IB	Phase B current	amps	only drivers
IC	Phase C current	amps	only drivers
R	Motor resistance	ohms	only drivers
RA	Phase A resistance	ohms	only drivers
RB	Phase B resistance	ohms	only drivers
RC	Phase C resistance	ohms	only drivers
T	Board temperature	° C	controllers and drivers
RT	Power supply temperature	° C	only controllers

Examples:

Command: `15:?MEAS VCC`

Answer: `15:?MEAS 80.1`

Command: `15:?MEAS T`

Answer: `15:?MEAS 24`

?MEMORY

Query available memory

Syntax:

<board_addr>:?MEMORY

Answer:

<board_addr>:?MEMORY <totalMemory> <freeMemory> <maxFreeBlock>

Description:

Returns the amount of total user memory <totalMemory>, unused memory <freeMemory> and the size of biggest available memory block <maxFreeBlock>. All the three quantities are returned in bytes.

Examples:

Command: `15:?MEMORY`

Answer: `15:?MEMORY ??? ??? ???`

MODE / ?MODE

Set/query board or system mode

Syntax:

MODE { OPER | PROG | TEST } **(system command)**

Description:

Changes the mode of the IcePAP system to operation (*OPER*), firmware reprogramming (*PROG*) or factory test (*TEST*) modes.

In normal operation, the system must be always set in mode *OPER*.

Syntax:

Answer:

?MODE { OPER | PROG | TEST } **(system answer)**

or

<board_addr>:?MODE { CONFIG | OPER | PROG | TEST | FAIL } **(board answer)**

Description:

Returns the current mode of the system or the specific mode of one of the boards (controllers or drivers). In normal conditions, all the boards should return the same mode than the system.

If a particular driver is switched into configuration mode (see CONFIG command), the returned mode is CONFIG for that driver (note that CONFIG is not a system mode selectable by the MODE command).

If a non recoverable internal hardware error happens in a particular board, that board switches FAIL mode.

Examples:

```
Command:    ?MODE
Answer:     ?MODE OPER
Command:    25:?MODE
Answer:     25:?MODE OPER
Command:    25:CONFIG
Command:    25:?MODE
Answer:     25:?MODE CONFIG
Command:    ?MODE
Answer:     ?MODE OPER
```

MOVE

Start absolute movement

Syntax:

<board_addr>:MOVE <absolutePos> **(board command)**

or

MOVE [GROUP] <axis1> <absPos1> … <axisN> <absPosN> **(system command)**

Description:

Performs an absolute movement on the specified axis or axes.

If the GROUP keyword is used with the MOVE system command, the set of axes is managed as a group by the system controller as described in 2.2.2.

Examples:

Command: `115:MOVE 4000`

Command: `MOVE 115 4000`

Command: `MOVE 31 250000 32 -29888 33 250000`

NAME / ?NAME

Set/query board name

Syntax:

<driver_addr>:NAME <driverName>

Description:

Sets the internal board name to the ASCII string <driverName>. This name is only used for identification purposes and user convenience.

The maximum length is 20 [TODO: CHECK] characters .

If the name is locked (configuration parameter NAMELOCK = YES) , this command will not have any effect.

Syntax:

<driver_addr>:?NAME

Answer:

<driver_addr>:?NAME <driverName>

Description:

Returns the board name string.

Examples:

Command:	`#11:NAME phi`
Answer:	`11:NAME OK`
Command:	`12:?NAME`
Answer:	`12:?NAME th`
Command:	`#12:NAME tth`
Answer:	`12:NAME ERROR The name of this board is locked`

NOECHO

Cancel echo mode

Syntax:

NOECHO **(system command)**

or

<board_addr>:NOECHO **(board command)**

Description:

Switches the echo mode off. See the ECHO command for more details. Only applies for serial line communication.

Example:

Command: `NOECHO`

Command: `2:NOECHO`

PCLOOP / ?PCLOOP

Set/query current position closed loop mode

Syntax:

<driver_addr>:PCLOOP {ON | OFF}

Description:

Activates/deactivates the position closed loop.

In order to activate the position closed loop, a target encoder must be configured (configuration parameter TGTENC must be different from NONE), and the difference between the position values in tgtenc and indexer must be inside the range +/- PCLERROR(see configuration parameters

---

Syntax:

<driver_addr>:?PCLOOP

Answer:

<driver_addr>:?PCLOOP {ON | OFF}

Description:

Returns the current position closed loop mode.

Examples:

Command: `15:PCLOOP ON`

Command: `15:?PCLOOP`

Answer: `15:?PCLOOP ON`

POS / ?POS

Set/query axis position in axis units

Syntax:

<board_addr>:POS [*pos_sel*] <posVal> **(board command)**

or

POS [*pos_sel*] <axis1> <posVal1> … <axisN> <posValN> **(system command)**

Description:

Loads the position registers in the specified boards with the <posVal> values. The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	
INDEXER	
POSERR	
SHFTENC *	
TGTENC *	
ENCIN *	
INPOS *	
ABSENC *	
MOTOR *	

* Only valid for driver boards

If position is not specified, the value is loaded as axis position

---

Syntax:

<board_addr>:?POS [*pos_sel*] **(board query)**

or

?POS [*pos_sel*] <axis1> <axis2> … <axisN> **(system query)**

Answer:

<board_addr>:?POS <posVal> **(board answer)**

or

<board_addr>:?POS <posVal1> <posVal2> … <posValN> **(system answer)**

Description:

Returns the current signal source used as axis indexer.

Examples:

Command: `115:POS AXIS 500`

Command: `115:POS INDEXER -3000`

Command: `115:?POS`

Answer: `115:?POS 500`

Command: `?POS INDEXER 5 115`

Answer: `?POS 13467895 -3000`

?POST

Query power-on selftest results

Syntax:

<board_addr>:?POST

Answer:

<board_addr>:?POST <testresultMask>

Description:

Returns the result of the power-on self tests as an binary mask <testresultMask>. A bit set to one in the mask indicates that a particular test has failed. If no tests failed during the power-on sequence, this query returns zero.

The meaning of the individual bits in <testresultMask> is summarised in the following table:

bit	*subsystem under test*	*applies to:*
0x01	external RAM	controllers and drivers
0x02	non volatile FRAM	controllers and drivers
0x04	internal 1-wire bus	controllers and drivers
0x08	ADC	only drivers
0x10	FPGA	controllers and drivers
0x20	external 1-wire bus	only controllers
0x40	CANbus	only controllers

Examples:

Command: `115:?POST`

Answer: `115:?POST 0`

POWER / ?POWER

Set/query motor power state

Syntax:

<driver_addr>:POWER [{ON | OFF}]

Description:

Switches on or off the motor power in a driver board.

Syntax:

<driver_addr>:?POWER

Answer:

<driver_addr>:?POWER {ON | OFF}

Description:

Returns the power state of the driver board.

Examples:

Command: `115:POWER OFF`

Answer: `115:POWER OFF`

*PROG / PROG / ?PROG

Firmware programming

Syntax:

***PROG {NONE | <bAddr> | DRIVERS | CONTROLLERS | ALL} [FORCE] [SAVE]**

PROG { <bAddr> | DRIVERS | CONTROLLERS | ALL} [FORCE]

Description:

The *PROG command reprograms the components of the IcePAP system by using firmware code that is transferred as a binary data block (see xxx). If the SAVE flag is used, the firmware code is stored in the non volatile FLASH memory of the system master board.

A mandatory parameter specifies the components to program, that can be either the components in the board with address <bAddr>, in all the driver boards (DRIVERS), in all controller boards (CONTROLLERS) or in both (ALL). If the parameter is set to NONE, no components are programmed, but the *PROG command can be use to store the firmware code in the system if the SAVE flag is used.

If one of the components in the system is already programmed with the same version of firmware, the programming operation for that specific component is skipped. This behaviour changes if the FORCE flag is used. In that case the components in the selected boards are always reprogrammed regardless of their current firmware version.

The *PROG command initiates the internal programming procedure and completes successfully if it started succesfully. The actual progress and the final success of the programming operation can be monitored by means of the ?PROG query.

The PROG command works in the same way than *PROG but uses the firmware code that was previously stored in the non volatile FLASH memory of the system master board by a previous *PROG SAVE commad.

Syntax:

?PROG

Answer:

?PROG [{OFF | ACTIVE <progress> | DONE | ERROR}]

Description:

Returns the state of firmware programming operations.

Examples:

REPORT / ?REPORT

Set/query asynchronous report settings

Syntax:

REPORT { ON | OFF } [<firstRack> <lastRack> [ ]]

Description:

The REPORT command allows to activate (ON) or deactivate (OFF) the asynchronous reporting feature on the current communication port. When asynchronous reporting is active in a particular port (RS232 serial line or TCP socket), the IcePAP master controller sends binary data blocks containing status and position information through that port to the listening device, usually the host computer. The binary blocks contain status and position information as the values returned by the ?FSTATUS and ?FPOS queries.

The data blocks contain the status as well as the axis and indexer positions for all the boards in the selected racks. The data block is sent whenever the data in the system master board changes or after a time interval of <maxPeriod> seconds.

The block includes data from all the boards in the racks from <firstRack> to <lastRack> that must be numbers from 0 to 15.

The format of the binary blocks is the following:

[TODO: explain block format and structure]

Syntax:

?REPORT

Answer:

?REPORT { ON | OFF } <firstRack> <lastRack> <maxPeriod>

Description:

Returns the status and range of the asynchronous status reporting feature.

Examples:

RESET

System or rack reset

Syntax:

RESET [<rackNumber>]

Description:

Resets the given rack or the whole system if no parameter is added.

Examples:

Command: `RESET`

Command: `RESET 8`

?RID

Query rack hardware identification string

Syntax:

?RID [<rackNumber1> <rackNumber2> ... <rackNumberN>]

Answer:

?RID <hwIDstring1> <hwIDstring2> ... <hwIDstringN>

Description:

The ?RID query returns the hardware identification strings of the racks specified by the list of rack numbers. If the query is issued with no parameters, it returns the identification strings of the rack 0 (the one hosting the system master controller).

Examples:

```
Command:    ?RID 0
Answer:     ?RID XXXX.XXXX.XXXX
Command:    ?RID
Answer:     ?RID XXXX.XXXX.XXXX     //rack 0 ID string
Command:    ?RID 5 6
Answer:     ?RID YYYY.YYYY.YYYY ZZZZ.ZZZZ.ZZZZ
```

RFPROG

Factory firmware programming

Syntax:

RFPROG [<rackNumber1> <rackNumber2> ... <rackNumberN>]

Description:

The RFPROG command is intended to reload the firmware in IcePAP driver boards that are not responsive or that have never been programmed (i.e after manufacturing). It cannot be used to reload firmware in controller boards and should not be used in normal operation instead of the PROG command.

RFPROG initiates the programming procedure of all the driver boards in the racks specified in the command line. If the command is issued with no parameters, it initiates the programming of the drivers boards in all the racks present in the system.

The system uses the firmware code that was stored in the non volatile FLASH memory of the system master board by a previous *PROG SAVE command.

The progress of the programming procedure can be followed by means of the ?PROG query.

Examples:

```
Command:    MODE PROG
Command:    *PROG NONE SAVE
            [firmware as binary data block]
Command:    RFPROG
                        (wait some time)
Command:    ?PROG
Answer:     ?PROG ACTIVE 37%
                        (wait for full reprogramming)
Command:    ?PROG
Answer:     ?PROG DONE
```

RMOVE

Start relative movement

Syntax:

<board_addr>:RMOVE <relativePos> **(board command)**

or

RMOVE <axis1> <relativePos1> … <axisN> <relativePosN> **(system command)**

Description:

Performs a relative movement on the specified axis or axes.

If the GROUP keyword is used with the RMOVE system command, the set of axes is managed as a group by the system controller as described in 2.2.2.

Examples:

```
Command:    115:?POS
Answer:     115:?POS 5000
Command:    115:RMOVE -7000
Command:    115:?POS
Answer:     115:?POS -2000

Command:    RMOVE 115 -7000
Command:    RMOVE 31 250000 32 -29888 33 250000
```

?RTEMP

Query rack temperatures

Syntax:

?RTEMP [<rackNumber1> <rackNumber2> ... <rackNumberN>]

Answer:

?RTEMP <rackTemp1> <rackTemp2> ... <rackTempN>

Description:

The ?RTEMP query returns the temperature of the main power supply of the racks specified by the list of rack numbers. If the query is issued with no parameters, it returns the temperatures of the rack with lowest number (the one hosting the system master controller).

Examples:

Command:	`?RTEMP 0`
Answer:	`?RTEMP 35`
Command:	`?RTEMP`
Answer:	`?RTEMP 35`
Command:	`?RTEMP 0 2 5`
Answer:	`?RTEMP 35 32 31`

SRCH

Start signal search sequence

Syntax:

<driver_addr>:SRCH <signal> [<edgetype> <srchdir>] **(driver command)**

With:

<signal> = {Lim- | Lim+ | Home | EncAux | InpAux}

<edgetype> = {POSEDGE | NEGEDGE}

<srchdir> = {+1 | -1}

Description:

Starts a signal search sequence.

The **<signal>** parameter is mandatory.

<edgetype> and **<srchdir>** parameters are ignored if the **<signal>** parameter is either lim- or lim+, otherwise these parameters have to be specified.

The search is done in the direction specified by **<srchdir>**. Positive and negative directions are selected by the values +1 and -1 respectively.

<edgetype> corresponds to the transition of the signal that defines the reference to search, posedge being a transition from inactive to active.

If the **<signal>** parameter is lim- or lim+, the direction and edge type will be selected automatically in order to start the search towards the specified limit switch position.

Examples:

```
Command:    16:?SRCHSTAT
Answer:     16:?SRCHSTAT NOTFOUND 0
Command:    16:SRCH HOME POSEDGE +1
Command:    16:?SRCHSTAT
Answer:     16:?SRCHSTAT MOVING +1
Command:    16:?SRCHSTAT FOUND +1
Answer:     16:SRCH LIM-
[. . .]
Command:    16:?SRCHSTAT
Answer:     16:?SRCHSTAT FOUND -1
```

?SRCHENC

Query the found signal search position in encoder steps

Syntax:

<board_addr>:?SRCHENC [*pos_sel*] **(driver command)**

Description:

Query the position registers latched when the signal event arrived during a signal search command. The answer is a number of encoder steps, the resolution being the one configured for the selected encoder.

If no signal search latch event happened after the last search, an error is issued

The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	
INDEXER	
POSERR	
SHFTENC *	
TGTENC *	
ENCIN *	
INPOS *	
ABSENC *	
MOTOR *	

* Only valid for driver boards

If pos_sel is not specified, the value returned is axis position at signal latch event.

Examples:

Command: `16:?SRCHPOS`

Answer: `16:?SRCHPOS ERROR Last home search was not successful`

Command: `16:SRCH LIM+`

Command: `16:?SRCHSTAT`

Answer: `16:?SRCHSTAT FOUND +1`

Command: `16:?SRCHENC`

Answer: `16:?SRCHENC 12345`

Command: `16:?SRCHENC TGTENC`

Answer: `16:?SRCHENC 24700`

?SRCHPOS

Query the found home position in axis units

Syntax:

<board_addr>:?SRCHPOS [*pos_sel*] **(driver command)**

Description:

Query the position registers latched latched when the signal event arrived during a signal search command.

The answer is a number of steps in axis units.

If no signal search latch event happened after the last search, an error is issued

The specific register is selected by the optional parameter *pos_sel*, that must be one of the following values:

pos_sel	*Position register*
AXIS	
INDEXER	
POSERR	
SHFTENC *	
TGTENC *	
ENCIN *	
INPOS *	
ABSENC *	
MOTOR *	

* Only valid for driver boards

If pos_sel is not specified, the value returned is axis position at signal latch event.

Examples:

Command: `16:?SRCHPOS`

Answer: `16:?SRCHPOS ERROR Last home search was not successful`

Command: `16:SRCH LIM1`

Command: `16:?SRCHSTAT`

Answer: `16:?SRCHSTAT FOUND +1`

Command: `16:?SRCHPOS`

Answer: `16:?SRCHPOS 12345`

Command: `16:?SRCHPOS TGTENC`

Answer: `16:?SRCHPOS 12350`

?SRCHSTAT

Query signal search status

Syntax:

<driver_addr>:?SRCHSTAT **(driver command)**

Answer:

<driver _addr>:?HOMESTAT {MOVING | FOUND | NOTFOUND} {+1| 0 |-1}(driver answer)

Description:

Query information about the ongoing or previous signal search sequence. If the sequence is in progress, ?HOMESTAT returns MOVING as hStatus keyword. If the sequence is finished, the query returns either FOUND or NOTFOUND depending on whether the home search succeeded or not.

The numeric value hDirection after the hStatus keyword, represents the direction of motion, -1 or +1, either during the search sequence or when the search is successfully completed. If the homing sequence fails, the returned hDirection value is 0.

Examples:

Command: `16:?SRCHSTAT`

Answer: `16:?SRCHSTAT NOTFOUND 0`

Command: `16:SRCH LIM+`

Command: `16:?SRCHSTAT`

Answer: `16:?SRCHSTAT MOVING +1`

Command: `16:?SRCHSTAT`

Answer: `16:?SRCHSTAT FOUND +1`

?STATUS

Query board status

Syntax:

<board_addr>:?STATUS **(board query)**

or

?STATUS <axis1> <axis2> … <axisN> **(system query)**

Answer:

<board_addr>:?STATUS <statusReg> **(board answer)**

or

?STATUS <statusReg1> <statusReg2> … <statusRegN> **(system answer)**

Description:

Returns the current status words of the specified boards as 32-bit values in C-like hexadecimal notation.

The system query ?STATUS can be used to return the status of any number of boards in the system. In that case the status value is sampled simultaneously in all the boards.

Note that in the cases of very frequent status polling, the ?FSTATUS query may be preferred to ?STATUS as ?FSTATUS returns values stored in the system controller and therefore suffers from shorter execution latency. ?STATUS on the other hand returns the status information stored in the boards and therefore guarantees more up to date values.

Example:

Command: `53:?STATUS`

Answer: `53:?STATUS 0x00000003`

Command: `?STATUS 80 83 85`

Answer: `?STATUS 0x002c0403 0x00200403 0x00000403`

STOP

Stop movement

Syntax:

<board_addr>:STOP **(board command)**

or

STOP [<axis1> <axis2> … <axisN>] **(system command)**

Description:

The STOP command finalises the movement in the given axes with a normal deceleration ramp.

If any of the axis is included in one of the currently active axis groups (see 2.2.2), all the other axes of the corresponding group are also stopped.

For security reasons, if the STOP system command fails or any error happens, all the boards in the system are instructed to stop their movements.

Examples:

Command: `10:STOP`

Command: `STOP` // stop all movements

Command: `#STOP 30 33 42`

Answer:
```
STOP ERROR All axes stopped, cause in axis 33:
Board is not present in the system
```

Command: `#STOP 30 rrt 42`

Answer:
```
STOP ERROR All axes stopped, cause: Wrong
parameter(s)
```

?SYSSTAT

Query system configuration

Syntax:

?SYSSTAT [<rackNumber>]

Answer:

?SYSSTAT <rackPresenceMask>

or

?SYSSTAT <driverPresenceMask> <driverAliveMask>

Description:

By default, with no parameter, the SYSSTAT query returns a 16 bit mask that represents the list of racks present in the system. Every bit in the <rackPresenceMask> mask indicates if the corresponding rack (0 to 15) has been found in the system.

If the SYSSTAT command is issued with a valid rack number (0 to 15) as parameter, it returns two 8 bit values that indicates the drivers found in the rack and which of them are responsive. Every bit of each mask correspond to one of the drivers (1 to 8) within the rack. The bits in <driverPresenceMask> that are set to 1, indicate which driver boards are plugged in the rack. The bits in <driverAliveMask> indicates which drivers are responsive and communicate with the system master board.

In normal conditions <driverPresenceMask> and <driverAliveMask> are identical.

Example:

Command:	`?SYSSTAT`
Answer:	`0x004F`
Command:	`?SYSSTAT 8`
Answer:	`0x13 0x13`

?TIME

Query the board running time

Syntax:

<board_addr>:?TIME

Answer:

<board_addr>:?TIME ***time_string***

Description:

Returns the time elapsed since the board processor start execution or was reset. The time is return as an ASCII string.

Examples:

Command: `115:?TIME`

Answer: `115:?TIME 1623hrs 31min 14sec`

VELOCITY / ?VELOCITY

Set/query programmed axis velocity

Syntax:

<board_addr>:VELOCITY [<velocity>] **(board command)**

or

VELOCITY <axis1> <velocity1> … <axisN> <velocityN> **(system command)**

Description:

Sets the velocity for the corresponding axis to the <velocity> values in steps per second. The actual acceleration for each axis is maintained to the previous value, and the acceleration time is internally recalculated (see ?ACCTIME query).

If no value is specified, the velocity is set to the default value.

Syntax:

<board_addr>:?VELOCITY **(board command)**

or

?VELOCITY <axis1> <axis2> … <axisN> **(system command)**

Answer:

<board_addr>:?VELOCITY <velocity> **(board answer)**

or

?VELOCITY <velocity1> <velocity2> … <velocity1> **(system answer)**

Description:

Returns the current velocity in steps per second.

Examples:

?VER

Query firmware version information

Syntax:

?VER [<verModule>] **(system command)**

or

<board_addr>:?VER [<verModule>] **(board command)**

Answer:

?VER <verModule> <verNumber> **(system answer)**

or

<board_addr>:?VER <verModule> <verNumber> **(board answer)**

Description:

Returns the version number XX.YY of the firmware.

module	*...*	*...*
SYSTEM		all cases
CONTROLLER		all cases
DRIVER		all cases
DSP		controllers and drivers
FPGA		controllers and drivers
PCB		controllers and drivers
IO		drivers
INFO		all cases

The ?VER INFO query returns a multiline answer with the version numbers of all the modules.

Example:

Command: ?VER

Answer: ...

?VSTATUS

Query verbose board status

Syntax:

<board_addr>:?VSTATUS

Answer:

<board_addr>:?VSTATUS <statusReg>

Description:

Returns the board status as as multiline verbose answer. The status information is the same returned by ?STATUS and ?FSTATUS, but the various status bits and fields are presented and identified separately.

The purpose of ?VSTATUS is to be used to assist application debugging and it is not intended to be used in normal operation.

Example:

Command: `12:?STATUS`

Answer: `12:?STATUS 0x00000003`

Command: `12:?VSTATUS`

Answer:

```
12:?VSTATUS $
0x00000000
  .......1 - 1: Board is present in the system
  .......2 - 1: Board is alive
  .......C - 2: Board mode is OPER
  ......7. - 2: Motor power is ON
  .....18. - 2: Indexer source is INTERNAL
  .....2.. - 1: Board is READY
  .....4.. - 0: Motor is not moving
[...]
  ...4.... - 0: Limit+ signal is not active
  ...8.... - 0: Limit- signal is not active
  ..1..... - 0: Home signal is not LOW
  ..2..... - 1: Aux 5V power supply is ON
  ..4..... - 0: Firmware versions are consistent
  FF...... - 0: Info value is 0
$
```

?WARNING

Query board warnings

Syntax:

<board_addr>:?WARNING

Answer:

<board_addr>:?WARNING NONE

or

<controller_addr>:?WARNING [<temperatureWarning>]

or

<driver_addr>:?WARNING $

[<temperatureWarning>]

[<ssiWarning>]

[<externalWarning>]

$

Description:

Returns a list of strings describing warning conditions that happened since the last ?WARNING query was received by the board. If no warning conditions happened, the query returns no strings.

The warning strings and conditions are cleared immediately after the query is executed.

Examples:

Command: `115:?WARNING`

Answer:
```
115:?WARNING $
blah, blah
$
```

Command: `115:?WARNING`

Answer: `115:?WARNING NONE`

WTEMP / ?WTEMP

Set/query warning temperature

Syntax:

<board_addr>:WTEMP <warningTemp>

Description:

Sets the temperature threshold used by the board to generate warning conditions to the value <warningTemp> in degrees Celsius.

Syntax:

<board_addr>:?WTEMP

Answer:

<board_addr>:?WTEMP <warningTemp>

Description:

Returns the warning temperature threshold in degrees Celsius.

Examples:

Command: `11:?WTEMP`

Answer: `11:?WTEMP 40`

Command: `12:WTEMP 35.5`

5.2. IcePAP command quick reference

BOARD COMMANDS

BOARD CONFIGURATION and IDENTIFICATION

<board_addr>:?ACTIVE Query activation status
<board_addr>:?MODE Query board mode
<board_addr>:?STATUS Query board status
<board_addr>:?VSTATUS Query verbose board status
<board_addr>:?ALARM Query board alarm message
<board_addr>:?WARNING Query board warnings
<board_addr>:WTEMP <warningTemp> **<board_addr>:?WTEMP** Set/query warning temperature
<driver_addr>:CONFIG [<confID>] **<driver_addr>:?CONFIG** Manage configuration mode
<driver_addr>:CFG <configPar> <configVal> **<driver_addr>:CFG {DEFAULT | EXPERT}** **<driver_addr>:?CFG {[<configPar>] | EXPERT}** Set/query configuration parameters
<driver_addr>:?CFGINFO [<configPar>] Query configuration parameter info
<board_addr>:?VER [<verModule>] Query board version information
<board_addr>:NAME <boardName> **<board_addr>:?NAME** Set/query board name
<board_addr>:?ID [{HW | SN}] Query board identification
<board_addr>:?POST Query power-on selftest results

POWER AND MOTION CONTROL

<driver_addr>:POWER [{ON | OFF}] **<driver_addr>:?POWER** Set/query motor power state
<driver_addr>:AUXPS [{ON | OFF}] **<driver_addr>:?AUXPS** Set/query auxiliary power supply state
<board_addr>:?MEAS {VCC | VM | IM | IA | IB | IC | T | RT} Query measured value
<board_addr>:POS [*pos_sel*] <posVal> **<board_addr>:?POS [*pos_sel*]** Set/query axis position in axis units
<board_addr>:ENC [*pos_sel*] <posVal> **<board_addr>:?ENC [*pos_sel*]** Set/query axis position in encoder steps
<driver_addr>:?HOMESTAT Query home search status
<driver_addr>:?HOMEPOS [*pos_sel*] Query the found home position in axis units
<driver_addr>:?HOMEENC [*pos_sel*] Query the found home position in encoder steps
<board_addr>:VELOCITY [<velocity>] **<board_addr>:?VELOCITY** Set/query programmed axis velocity
<board_addr>:ACCTIME [<accTime>] **<board_addr>:?ACCTIME** Set/query acceleration time
<driver_addr>:PCLOOP {ON | OFF} **<driver_addr>:?PCLOOP** Set/query current position closed loop mode

<driver_addr>:ESYNC Synchronise internal position registers
<board_addr>:RMOVE <absolutePos> Start relative movement
<board_addr>:JOG <signedVelocity> **<board_addr>:?JOG** Set/query jog velocity
<driver_addr>:HOME [{+1|0|-1}] Start home signal search sequence
<driver_addr>:CMOVE <absolutePos> Start relative movement in configuration mode
<driver_addr>:CJOG <signedVelocity> Set jog velocity in configuration mode
<board_addr>:STOP Stop movement
<board_addr>:ABORT Abort movement

INPUT/OUTPUT

<driver_addr>:INDEXER [{INTERNAL | SYNC | INPOS | ENCIN}] **<driver_addr>:?INDEXER** Set/query indexer signal source
<driver_addr>:INFOA [*signal_source* [{NORMAL | INVERTED}]] **<driver_addr>:INFOB [*signal_source* [{NORMAL | INVERTED}]]** **<driver_addr>:INFOC [*signal_source* [{NORMAL | INVERTED}]]** **<driver_addr>:?INFOA** **<driver_addr>:?INFOB** **<driver_addr>:?INFOC** Set/query Info signal source and polarity

COMMUNICATION and ERROR MANAGEMENT

<board_addr>:?HELP Query list of available board commands
<board_addr>:?ERRMSG Local query of last command error message
<board_addr>:?FERRMSG Query first error message
<board_addr>:BLINK <blinkTime> **<board_addr>:?BLINK** Set/query remaining blinking time
<board_addr>:?TIME Query running time
<board_addr>:DEBUG <debugLevel> **<board_addr>:?DEBUG** Set/query debug level
<board_addr>:ECHO Select echo mode
<board_addr>:NOECHO Cancel echo mode
<board_addr>:?MEMORY Query available memory
<board_addr>:?ADDR Query board address

SYSTEM COMMANDS

SYSTEM CONFIGURATION and IDENTIFICATION

MODE {OPER | PROG | TEST} **?MODE** Set/query system mode
?SYSSTAT [<rackNumber>] Query system configuration
?STATUS <axis1> <axis2> … <axisN> Query multiple board status
?FSTATUS [<axis1> <axis2> … <axisN>] Fast query of multiple board status
REPORT {ON | OFF} [<firstRack> <lastRack> [<maxPeriod>]] **?REPORT** Set/query asynchronous report settings
?VER [<verModule>] Query system firmware version information
?RID [<rackNumber1> <rackNumber2> … <rackNumberN>] Query rack identification string
?RTEMP [<rackNumber1> <rackNumber2> … <rackNumberN>] Query rack temperatures
***PROG {NONE | <bAddr> | DRIVERS | CONTROLLERS | ALL} [FORCE] [SAVE]** **PROG {<bAddr> | DRIVERS | CONTROLLERS | ALL} [FORCE]** **?PROG** Firmware programming
RFPROG [<rackNumber1> <rackNumber2> … <rackNumberN>] Factory firmware programming
RESET [<rackNumber>] System or rack reset

MOTION CONTROL

POS [*pos_sel*] <axis1> <posVal1> … <axisN> <posValN> **?POS [*pos_sel*] <axis1> <axis2> … <axisN>** Set/query multiple axis position in axis units
ENC [*pos_sel*] <axis1> <posVal1> … <axisN> <posValN> **?ENC [*pos_sel*] <axis1> <axis2> … <axisN>** Set/query multiple axis position in encoder steps
?FPOS [*pos_sel*] <axis1> <axis2> … <axisN> Fast query of multiple board positions
?HOMESTAT <axis1> … <axisN> Query multiple axis home search status
?HOMEPOS [*pos_sel*] <axis1> … <axisN> Query the found multiple axis home position in axis units
?HOMEENC [*pos_sel*] <axis1> … <axisN> Query the found multiple axis home position in encoder steps
?VELOCITY <axis1> <velocityN> … <axisN> <velocityN> **?VELOCITY <axis1> <axis2> … <axisN>** Set/query programmed multiple axis velocity
ACCTIME <axis1> <accTime1> … <axisN> <accTimeN> **?ACCTIME <axis1> <axis2> … <axisN>** Set/query acceleration time
MOVE [GROUP] <axis1> <absolutePos1> … <axisN> <absolutePosN> Start multiple axis absolute movement
RMOVE [GROUP] <axis1> <absolutePos1> … <axisN> <absolutePosN> Start multiple axis relative movement
JOG [GROUP] <axis1> <signedVel1> … <axisN> <signedVelN> **?JOG <axis1> <axis2> … <axisN>** Set/query multiple axis jog velocities
HOME [GROUP] <axis1> {+1|0|-1} … <axisN> {+1|0|-1} Start multiple axis home signal search sequence
STOP [<axis1> <axis2> … <axisN>] Stop multiple axis movement
ABORT [<axis1> <axis2> … <axisN>] Abort movement

?HELP	Query list of available commands
?ERRMSG	Query last command error message
ECHO	Select serial line echo
NOECHO	Cancel serial line echo