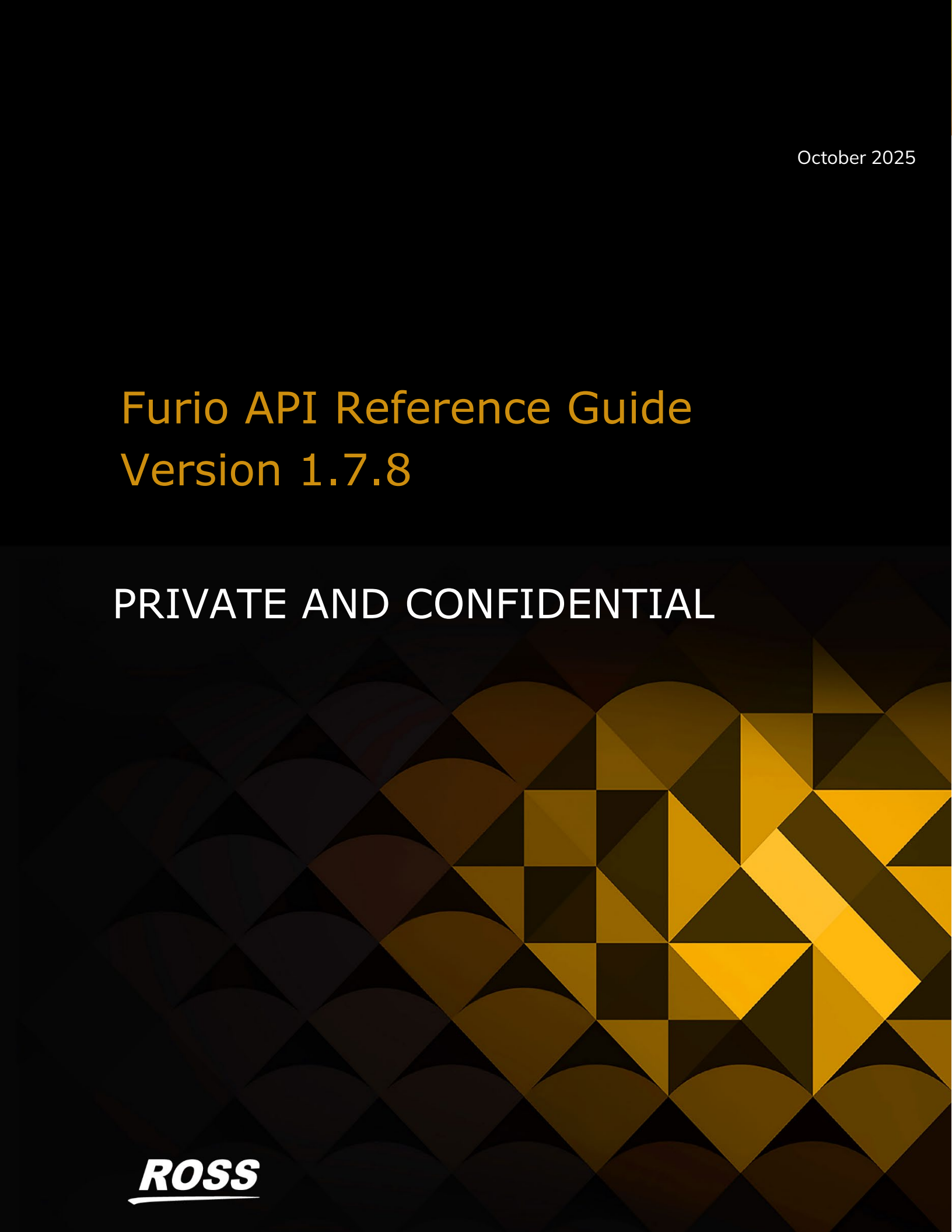
October 2025

Furio API Reference Guide
Version 1.7.8

PRIVATE AND CONFIDENTIAL



ROSS

Thank you for choosing Ross

You've made a great choice. We expect you will be very happy with your purchase of Ross Technology. Our mission is to:

1. Provide a Superior Customer Experience
   - offer the best product quality and support
2. Make Cool Practical Technology
   - develop great products that customers love

Ross has become well known for the Ross Video Code of Ethics. It guides our interactions and empowers our employees. I hope you enjoy reading it below.

David Ross

CEO, Ross Video

dross@rossvideo.com

Ross Video Code of Ethics

Any company is the sum total of the people that make things happen. At Ross, our employees are a special group. Our employees truly care about doing a great job and delivering a high quality customer experience every day. This code of ethics hangs on the wall of all Ross Video locations to guide our behavior:

1. We will always act in our customers' best interest.
2. We will do our best to understand our customers' requirements.
3. We will not ship crap.
4. We will be great to work with.
5. We will do something extra for our customers, as an apology, when something big goes wrong and it's our fault.
6. We will keep our promises.
7. We will treat the competition with respect.
8. We will cooperate with and help other friendly companies.
9. We will go above and beyond in times of crisis. If there's no one to authorize the required action in times of company or customer crisis - do what you know in your heart is right. (You may rent helicopters if necessary.)

Furio API Reference Guide

- Ross Part Number: 5100DR-014-1.7.8
- Publication Date: October 21, 2025.
  Printed in Canada.

Copyright

Patents

Ross Video products are protected by patent numbers US 7,034,886; US 7,508,455; US 7,602,446; US 7,802,802 B2; US 7,834,886; US 7,914,332; US 8,307,284; US 8,407,374 B2; US 8,499,019 B2; US 8,519,949 B2; US 8,743,292 B2; GB 2,419,119 B; GB 2,447,380 B. Other patents pending.

Company Address

ROSS®

Ross Video Limited
8 John Street
Iroquois, Ontario
Canada, K0E 1K0

Ross Video Incorporated
P.O. Box 880
Ogdensburg, New York
USA 13669-0880

General Business Office: (+1) 613 •652 •4886
Fax: (+1) 613 •652 •4425

Technical Support: (+1) 613 •652 •4886
After Hours Emergency: (+1) 613 •349 •0006

E-mail (Technical Support): techsupport@rossvideo.com
E-mail (General Information): solutions@rossvideo.com
Website: www.rossvideo.com

CONTENTS

Understanding the Furio API

The Furio API Interface is conceived as a simple text-based protocol operating over TCP/IP connections. Two different connections can be made to a robotic head that implements the Furio API:

- **Control Port:**

  Connections to this port provide a basic text-based command line interface that exposes the Furio functionality to the outside world. Commands can be issued in a form that is both suited to manual testing by service personnel and for integrating with other automation systems.

  The Furio device accepts multiple concurrent connections to the Control Port. The TCP port number varies depending on the type of robotic head:

  - Furio heads:
    10240
  - CamBots hosted through the Robotics Server:
    13000 + 2 x ConfigPosition **

  The Command Overview chapter deal with the commands that are supported over this connection.

- **Status Port:**

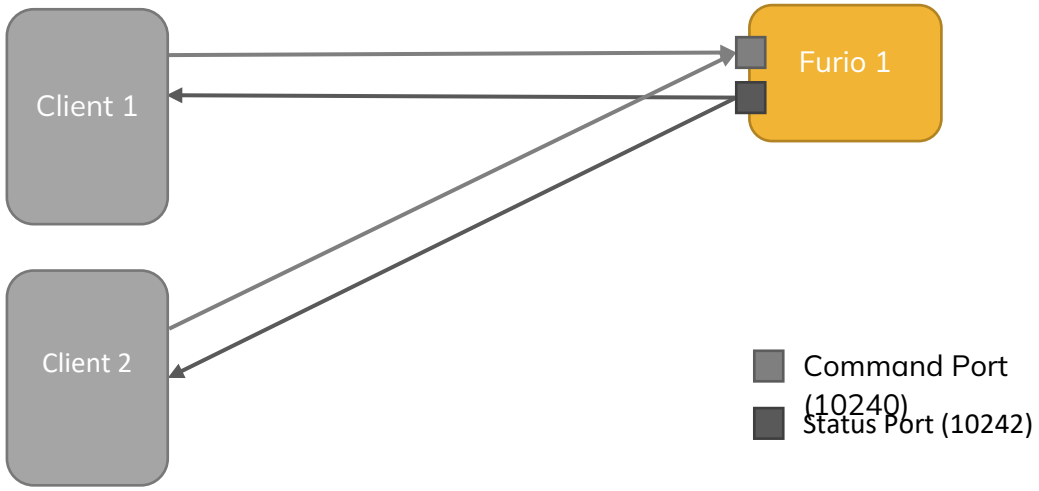
  Connections to this port provide a read only communications pathway over which the Furio head broadcasts information regarding its internal state. For example, if a new move or a new preset is created by a client using the Control Port then a message will be broadcast over this connection notifying all other clients.

  The Furio head accepts multiple concurrent connections to the Control Port. Again, the TCP port number varies depending on the type of robotic head:

  - Furio heads:
    10242
  - CamBots hosted through the Robotics Server:
    13001 + 2 x ConfigPosition**

** ConfigPosition is defined as the ordinal position of the device's <Robot> entry in the RoboticsServer.config file. ConfigPosition starts at 1.

Two Clients Connected to One Furio



Two Clients Connected to Two CamBots via Robotics Server

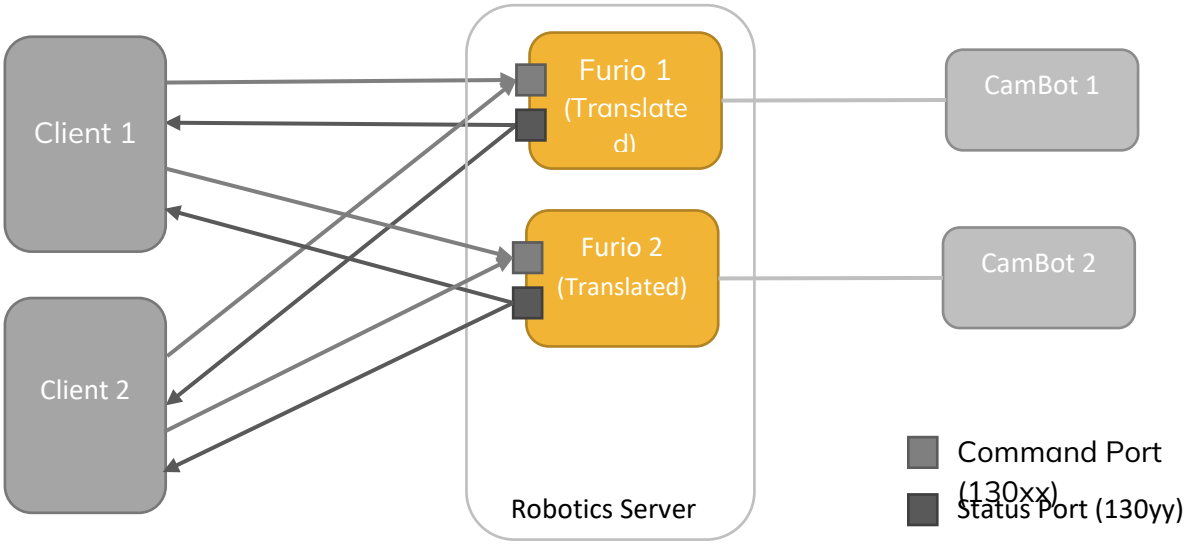
In the diagram below, if Furio 1 and Furio 2 were, respectively, the first and second entries in the Robotics Server config file, their Command and Status Port numbers would be:

13002: Furio 1 Command Port

13003: Furio 1 Status Port

13004: Furio 2 Command Port

13005: Furio 2 Status Port



The Robotic Server only listens for connections on its Command Port when it is connected to the corresponding CamBot.

Control Port Usage

ESTABLISHING A CONNECTION

After establishing a TCP/IP connection on the control port, the Furio head sends a prompt asking for a password. An automated client must login using auto/auto. The recommended login procedure for an automated client is as follows:

- Connect to the control port for the appropriate device type.
- Immediately send username 'Client', followed by a newline.
- Next send password 'Client', followed by a newline.
- Next send the SESSION command.
- Read and discard all input lines until a line containing the valid reply to the SESSION command is detected.
- The connection is now ready to accept commands.

Note: when we use the term **newline** in this document, we refer to a single carriage return character (`\r`), a single newline character (`\n`) or the combination of both (`\r\n`). This is to say that the Furio head accepts all these combinations. The Furio head uses `\r\n` in all replies, but it is recommended that the client software should also allow for the other options. Newlines are shown as ¶ in the code examples.

Note: The login auto/auto can also be used – but in that case only a limited number of commands will be available. Commands that are not supported will return ERR:COMMAND.

COMMAND STRUCTURE

The control port protocol is a simple synchronous client server dialogue. The client always initiates the action by sending a command. The client must then wait for the server to reply. Only when a complete reply has been received a new command may be sent.

It is recommended that the client implements a configurable timeout. If no complete reply has been received from the server within the specified timeout, then the client should close the connection (this could for example happen when the power to the Furio head has been cut). It can then go through the steps outlined in the previous paragraph to establish a new connection. It should continue doing this until a connection has been established successfully.

ENCODING AND HANDLING OF SPECIAL CHARACTERS

All communications is done using plain text in UTF-8 encoding. Some characters are forbidden inside command arguments because they have a special meaning, either in the command format itself or in the reply format. These characters need to be escaped. The escape mechanism used is a simple variation of the escape mechanism of the C language:

UTF-8 Character	Escaped Character
Newline	\n
Carriage return	\r
Tab	\t
\	\\
Space	\.
Comma	\k
{	\o
}	\c
=	\e
:	\d
;	\s

COMMAND FORMAT

Commands always have the same structure:

command subcommand [subcommand ...] argument1 argument2 ... argument n¶

The subcommands and the arguments are optional, depending on the actual command. (Sub)Commands and Arguments are separated by a single space character. A command is always terminated by a newline. The command and subcommand fields are case insensitive as well as some parameters. For example, when specifying an axis the argument is case insensitive (“`pan`”, “`PAN`”, “`Pan`”, etc are all valid). Other arguments like captions are stored case sensitive.

REPLY FORMAT

The Furio head acknowledges each command with a reply. A reply consists of one or more reply records. A reply record consists of a header and a body and is terminated by a newline. The body consists of a context value and a payload, which is dependent on the type of record

header{context:payload}¶

The header determines the type of reply record. There are 4 possible header types:

- ERR
- OK
- INFO
- EVT

ERR RECORD

An ERR record is used when an error occurred while executing or parsing the command. An ERR record always signifies the end of the reply and no further records will follow. The payload of an ERR record consists of an action and a reason:

ERR{context:action=reason}¶

In the special case where the original command was not recognized or not supported, the action will be omitted from the error record to give the format ERR {context:reason}. This format is used for COMMAND and NOFEATURE errors.

ERR{context: reason}

The meaning of the context field is command dependent. Typically, it reflects the unique ID of the data that the command operated on (e.g. Preset ID). When no relevant context exists, this is indicated by a context value of "*". The action is a unique identifier that reflects the original command that was issued by the client. The reason is an alphanumerical error code that gives more details about the error.

INFO RECORD

A record of type INFO is used to return information to the client. The Furio head can return zero or more INFO records depending on the command. The payload of an INFO record is a comma-separated list of name value pairs. Forbidden characters in the value fields are escaped in the same way as discussed earlier for the command arguments.

INFO{context:name=value,...}¶

To reduce the size of the network messages the server may omit name value pairs if the value happens to be equal to the default value for that property. Unless explicitly stated otherwise in the command description, default values are assumed as follows:

Data Type	Default Value
Number	0
Float	0.0
Boolean	False
String	Empty

This means that when a client expects to find a certain key value pair in an INFO record, but it is not present, it should not treat this as an error but should assume that the value for that key is simply the default.

OK RECORD

A record of type OK is used to signify the successful completion of a command. No more records will follow an OK record.

OK{context:action}¶

EVT RECORD

Records of type EVT are only used by the Status Port and will be discussed later in this document.

EVT{context:action}¶

AXES

Several commands require an axis as parameter. Allowed values depend on the configuration of the Furio head. Typically, they are: PAN, TILT, TRACK, LIFT, ZOOM, and FOCUS. On CamBots, PAN, TILT, XY, LIFT, ZOOM, and FOCUS are typical. Other axes may be added in the future.

UNITS OF MEASURE

For axis positions expressed in “real world units”:

- Millimeters are used to represent linear axis positions (LIFT, TRACK, XY).
- Degrees are used to represent rotational axis positions (PAN, TILT)
- Encoder counts are used to represent lens axis positions (ZOOM, FOCUS, IRIS).

Some commands represent axis positions in raw encoder/motor counts.

Examples

This section includes a few command and response examples.

For code examples in this document, the following typographical style conventions are used:

- **`Bold courier text`** denotes data sent by the client to the Furio head.
- *`Italic courier text`* denotes data sent by the Furio head to the client.
- Square brackets [ ] denote optional arguments.

EXAMPLE 1: NOP

The simplest command is the NOP command. This command doesn't do anything except reply an OK response from the server. Clients can use this command to check if the communications channel is (still) functional.

```
NOP¶
OK{*:NOP}¶
```

One can see that the server returns one record of type OK. The context in this case is '*'. The action field in the OK record reflects that this reply is in response to a NOP command.

EXAMPLE 2: CATEGORY LIST

The command CATEGORY LIST can be used to request a list of categories from the Furio head. There is this information to be sent from the Furio head to the Client. This is done using INFO fields.

```
CATEGORY LIST¶
INFO{8:CAPTION=Studio\.1}¶
INFO{10:CAPTION=News\.Show}¶
INFO{13:CAPTION=Talkshow\d\.Late\.Night}¶
OK{*:CATEGORYLIST}¶
```

In this case three INFO records are returned and one OK record. Each INFO record contains information about one single category. The context field is used to communicate the unique ID of the category. The payload section contains the attributes of a category, namely its caption. Note how the spaces and the colon in the category names are escaped.

More complete examples can be found at the end of the document.

Command Overview

This section of the document describes the various commands supported by the Furio system.

For code examples in this document, the following typographical style conventions are used:

- **`Bold courier text`** denotes data sent by the client to the Furio head.
- *`Italic courier text`* denotes data sent by the Furio head to the client.
- Square brackets [ ] denote optional arguments.

Axis Commands

AXIS INFO

The AXIS INFO command returns details of the specified Axis.

It is not supported by Robotics Server; use of AXIS STATUS is recommended instead.

Command:

```
AXIS INFO axis¶
```

Arguments:

```
axis: String: The axis to list.
```

Response:

```
INFO
   Context: String: Axis
   Name/Value Pairs:
         ENABLE: Enumeration: ENABLED, DISABLED
         INVERTED: Boolean
         MOTORPOSITION: Number: position in encoder counts
         POSITION: Float: position in real world units
         RATIO: Number
         DAMPING: Number
         ACCELDAMPING: Number
         DECELDAMPING: Number
         DYNAMICRATIO: Number (if configured)
OK
   Context: String: Axis
   Action: String: AXISINFO
```

Errors

```
NOAXISARG
```

NOTAXIS

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

AXIS INFO PAN¶

INFO{PAN:ENABLE=ENABLED,INVERTED=FALSE,MOTORPOSITION=124556,POSITION=51.257613,RATIO=50,DAMPING=50}

OK{PAN:AXISINFO}¶

AXIS LIST

The AXIS LIST command returns a list of all the supported axes.

Command:

AXIS LIST¶

Arguments:

None

Response:

```
INFO
   Context: String: Axis
   Name/Value Pairs:
         ENABLED: Enumeration: ENABLED, DISABLED
         INVERTED: Boolean
         HOMING: Enumeration: ERROR, HOMING, NA, HOMED, or NOTHOMED
OK
   Context: String: *
   Action: String: AXISLIST
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

AXIS LIST¶

INFO{XY:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NOTHOMED}

INFO{PAN:ENABLED=DISABLED,INVERTED=TRUE,HOMING=NOTHOMED}

INFO{TILT:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NOTHOMED}

INFO{LIFT:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NOTHOMED}

INFO{ZOOM:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NA}

INFO{FOCUS:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NA}

OK{:AXISLIST}¶*

AXIS SET DYNAMIC RATIO

The AXIS SET DYANMICRATIO command configures the strength of zoom var, that is, by what factor joystick demand is scaled back proportional to the zoom position. Not all axes support dynamic ratio.

Command:

AXIS SET DYNAMICRATIO axis¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

Response:

OK

Context: String: axis

Action: String: AXISSETDYNAMICRATIO

Errors

NOTAXIS

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED – if the axis dynamic ratio changes

Example:

AXIS SET DYNAMICRATIO PAN 100¶

OK{PAN:AXISSETDYNAMICRATIO}¶

EVT{42:AXISMODIFIED}

OK{PAN:AXISMODIFIED}

AXIS STATUS

The AXIS STATUS command queries the status of an axis.

Command:

AXIS STATUS axis¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

Response:

```
INFO (All axes including XY)
   Context: String: axis
   Name/Value Pairs:
         ENABLE: Enumeration: ENABLED, DISABLED
         INVERT: Enumeration: YES, NO
         MTRPOS: Float: position in real world units
               (see below for special XY axis position encoding)
         RATIO: Number
         ACCELDAMPING: Number
         DECELDAMPING: Number
         DYNAMICRATIO: Number (if configured)
         MODE: String – Legacy, recommend ignore.
         NKEYS: Number: Legacy, recommend ignore.
         HIGHLIM: Float: active limit; position in real world units
(or NONE)
         LOWLIM: Float: active limit; position in real world units
(or NONE)
         PERSISTENTHIGHSET: Boolean: whether persistent limit is
defined or using defaults (Since Furio 6.0b)
         PERSISTENTLOWSET:  Boolean: whether persistent limit is
defined or using defaults (Since Furio 6.0b)
         PERSISTENTHIGHLIM: Float: persistent limit; position
                                   in real world units (or NONE)
(Since Furio)
         PERSISTENTLOWLIM:  Float: persistent limit; position
                                   in real world units (or NONE)
(Since 1.7.0)
         TEMPORARYHIGHLIM:  Float: temporary limit; position
                                   in real world units (or NONE)
(Since 1.7.0)
         TEMPORARYLOWLIM:   Float: temporary limit; position
                                  in real world units (or NONE) (Since
1.7.0)
         HIGHLIMSOURCE: Enumeration: PERSISTENT, TEMPORARY; active
limit
                              source (Since Furio release 6.0b)
         LOWLIMSOURCE:  Enumeration: PERSISTENT, TEMPORARY; active
limit
                              source (Since Furio release 6.0b)
         HOMING: Enumeration: HOMED, NOTHOMED, HOMING, NA,
         EXTENDEDLIMITS: Enumeration: OK, BEYOND, NOTACTIVE; status
   of
               enforcement of extended limits.

               OK signals that extended XY limits (including legacy
   limits)
```

```
                are operating normally and enforced.

                BEYOND signals the pedestal is outside the operating
   zone
                defined by boundary(ies) and obstacles in the studio
   map,
                including the legacy XY limits.

                NOTACTIVE signals the studio map failed to load during
   boot.
                Note: When the value of  EXTENDEDLIMITS changes, the
   robot
                sends out an AXISMODIFIED event.
```

Two supplemental records returned for XY axis only:

```
   Context: String: XY
   Name/Value Pairs:
         COMPONENT: Enumeration: X, Y
         HIGHLIM: Float: active limit, position in real world units
                              (or NONE)
         LOWLIM: Float: active limit, position in real world units
                             (or NONE)
         PERSISTENTHIGHLIM: Float: persistent limit; position
                                  in real world units (or NONE)
(Since Furio 6.0a)
         PERSISTENTLOWLIM: Float: persistent limit; position
                                  in real world units (or NONE) (Since
Furio 6.0a)
         TEMPORARYHIGHLIM: Float: temporary limit; position
                                  in real world units (or NONE) (Since
Furio 6.0a)
         TEMPORARYLOWLIM:   Float: temporary limit; position
                                  in real world units (or NONE) (Since
Furio 6.0a)
         HIGHLIMSOURCE:     Enumeration: PERSISTENT, TEMPORARY;
                                  active limit source (Since 1.7.0)
         LOWLIMSOURCE:      Enumeration: PERSISTENT, TEMPORARY;
                                  active limit source (Since 1.7.0)
OK
   Context: String: axis
   Action: String: AXISSTATUS
```

Errors

```
NOTAXIS
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
AXISMODIFIED – XY axis: when the value of EXTENDEDLIMITS changes
```

Examples:

AXIS STATUS PAN¶

```
INFO{PAN:RATIO=100,DYNAMICRATIO=100,ACCELDAMPING=1,DECELDAMPING=1,
ENABLE=ENABLED,INVERT=NO,MODE,MTRPOS=23.0,HOMING=HOMED,HIGHLIM=85.83
1,
LOWLIM=-85.831,TEMPORARYHIGHLIM=178.000,TEMPORARYLOWLIM=-
178.000,PERSISTENTHIGHLIM=178.000,PERSISTENTLOWLIM=-178.000}¶
OK{PAN:AXISSTATUS}¶
```

AXIS STATUS XY¶

```
INFO{XY:RATIO=50,ACCELDAMPING=50,DECELDAMPING=50,ENABLE=DISABLED,
INVERT=NO,MODE,MTRPOS=90061084.61083,HOMING=NOTHOMED,HIGHLIM=NONE,
LOWLIM=NONE,TEMPORARYHIGHLIM=NONE,TEMPORARYLOWLIM=NONE,PERSISTENTHIG
HLIM=NONE,PERSISTENTLOWLIM=NONE}¶
INFO{XY:COMPONENT=X,HIGHLIM=NONE,LOWLIM=NONE,TEMPORARYHIGHLIM=178.00
0,TEMPORARYLOWLIM=-
178.000,PERSISTENTHIGHLIM=60.960,PERSISTENTLOWLIM=-60.960}¶
INFO{XY:COMPONENT=Y,HIGHLIM=NONE,LOWLIM=NONE,TEMPORARYHIGHLIM=178.00
0,TEMPORARYLOWLIM=-
178.000,PERSISTENTHIGHLIM=60.960,PERSISTENTLOWLIM=-60.960}¶
OK{XY:AXISSTATUS}¶
```

For the XY axis, the first INFO record lists all the details except the high and low limits. The second INFO record gives the high and low limits of the X axis, and the third INFO record gives the high and low limits of the Y axis.

Also, for the XY axis, the X position, Y position and pedestal rotation angle are encoded into the MTRPOS argument as follows.

Let:

XPOS: the X position in millimeters, range [-60960, 60960]

YPOS: the Y position in millimeters, range [-60960, 60960]

THETA: the ped angle in degrees, range [-360, 360]

Then:

*MTRPOS = (1000000 * THETA) + (XPOS + 60960) + ((YPOS + 60960)) / 100000)*

CLEAR FAULTS

The CLEAR FAULTS command is used to clear faults on an axis. The state of an axis can be checked with the AXIS STATUS command.

Command:

CLEARFAULTS axis¶

Arguments:

axis: String: Axis for which faults should be cleared.

Response:

OK

Context: String: Axis for which fault clearing was requested

Action: String: CLEARFAULTS

Errors

NOARG

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

CLEARFAULTS PAN¶

OK{PAN:CLEARFAULTS}¶

DISABLE

The DISABLE command is used to issue a disable request to an axis. Disabling the axis is done by a background process. An OK reply does not mean the axis is disabled. The state of an axis can be checked with the AXIS INFO command.

Command:

DISABLE axis¶

Arguments:

axis: String: Axis that should be disabled.

Response:

OK

Context: String: Axis that should be disabled

Action: String: DISABLE

Errors

NOARG

NOTSUPPORTED (CamBot)

ESTOP

Synchronous Status Events:

AXISMODIFIED – if the axis state changes

Asynchronous Status Events:

PRESETCANCELLED

MOVECUTCANCELLED

Example:

DISABLE PAN¶

OK{PAN:DISABLE}¶

EVT{42:AXISMODIFIED}
OK{PAN:AXISMODIFIED}

ENABLE

The ENABLE command is used to issue an enable request to an axis. Enabling the axis is done by a background process. An OK reply does not mean the axis is enabled. The state of an axis can be checked with the AXIS INFO command.

Command:

ENABLE axis¶

Arguments:

axis: String: Axis that should be enabled.

Response:

OK

Context: String: Axis that should be enabled

Action: String: ENABLE

Errors

NOARG

NOTAXIS (Furio)

NOTSUPPORTED (CamBot)

ESTOP

Synchronous Status Events:

AXISMODIFIED – if the axis state changes

Asynchronous Status Events:

PRESETCANCELLED

MOVECUTCANCELLED

Example:

ENABLE PAN¶

OK{PAN:ENABLE}¶

EVT{42:AXISMODIFIED}
OK{PAN:AXISMODIFIED}

HOME

The HOME command is used to align an axis with real-world encoders. After booting the Furio head, an axis needs to home to know at which physical position it is. This is done by relating to encoder values measuring real-world offsets (degrees, meters,...). Once homed this command does not do anything unless the “force” argument is given. The homing state of an axis can be verified with the AXIS STATUS command.

An axis with relative encoders may move during homing, while an axis with absolute encoders is unlikely to move.

A CamBot XY axis must be positioned over the target for the HOME XY command to succeed.

Command:

HOME [force] axis¶

Arguments:

force (optional): Constant: Force (re-)homing.

axis: String: Axis that should be homed.

Response:

OK

Context: String: Axis that should be homed

Action: String: HOME

Errors

NOTAXIS

ENABLE=DISABLED

ESTOP

Synchronous Status Events:

HOME – Homing has started.

Asynchronous Status Events:

None

Example:

HOME PAN¶

OK{PAN:HOME}¶

EVT{42:HOME}

OK{PAN:HOME}

INVERT

The INVERT command inverts the axis direction with respect to incoming joystick demand. The inversion state of an axis can be verified with the AXIS STATUS command.

Command:

INVERT axis¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

Response:

OK

Context: String: axis

Action: String: INVERT

Errors

NOTAXIS

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED

Example:

INVERT PAN 100¶

OK{PAN:INVERT}¶

EVT{42:AXISMODIFIED}

OK{PAN:AXISMODIFIED}

SET DAMPING

The SET DAMPING command configures the acceleration/deceleration applied to direct control demand. Argument damping is expressed as a percentage of the maximum joystick acceleration.

Command:

SET DAMPING axis damping¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

damping: Number: percentage value between 1 and 100

Response:

OK

Context: String: axis

Action: String: SET DAMPING

Errors

NOTAXIS

BADDAMPINGARG

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED – if the axis damping value changes

Example:

SET DAMPING PAN 100¶

OK{PAN:SETDAMPING}¶

EVT{42:AXISMODIFIED}
OK{PAN:AXISMODIFIED}

SET HIGH LIMIT

The SET HIGH LIMIT command is used to set and unset a temporary or persistent position limit for the 'upper' end of an axis.

- To set a temporary high limit, use Type=TEMPORARY. The command will toggle the high limit between the persistent limit and the current position.
- To set a persistent high limit, use Type=PERSISTENT. The command will toggle the high persistent limit between the high limit configured in the template and the current position. Setting persistent limits through the Furio API is not supported for the XY axis.

The active high limit can be verified with the AXIS STATUS command.

Temporary limits are lost when the robot is rebooted.

Command:

SET HIGHLIMIT axis [type]¶

Arguments:

axis: String: The axis to configure.
Type: String; Optional; value can be "PERSISTENT" or "TEMPORARY"
If Type param is omitted, it will default to temporary.
XY does not support <type> param

Response:

OK
Context: String: *
Action: String: SETHIGHLIMIT

Errors

LIMITEXCLUDESHOME

Synchronous Status Events:

None

Asynchronous Status Events:

LIMITSET/LIMITCLEARED

If high temporary limit is set, LIMITSET event will be sent to status port;

If high temporary limit is unset, LIMITCLEARED event will be sent to status port;

High Persistent limit set/unset, does not trigger a status event.

Example:

1) No Type parameter, to toggle axis High temporary limit:

SET HIGHLIMIT PAN¶

OK{PAN:SETHIGHLIMIT}¶

2) *Type set to persistent:*

SET HIGHLIMIT PAN PERSISTENT¶

OK{PAN:SETHIGHLIMIT}¶

** * Axis XY does not support persistent limit toggle through command line, it only can toggle temporary, cmd format is:*

SET HIGHLIMIT XY Y¶

SET HIGHLIMIT XY X¶

SET INPUT ANALOGVELOCITY

The SET INPUT ANALOGVELOCITY command configures joystick control for an axis in velocity mode. Joystick input send to the Furio head for an axis configured in this mode translates to a velocity. The input value is relative to the configured minimum and maximum value which correspond to the configured maximum joystick control velocity in either direction.

When the LINEAR keyword is present, the robot's velocity will linearly correlate to the joystick input. For finer control at lower commanded velocities, omit the LINEAR keyword.

Command:

SET INPUT ANALOGVELOCITY axis [input_scaling] initial_value MM minimum maximum¶

Arguments:

axis: String: The axis to configure.

input_scaling: String: LINEAR, CUBIC, QUINTIC, or EXPONENTIAL – default is LINEAR

initial_value: Number: Initial velocity.

MM: Constant: MM stands for Minimum/Maximum.

minimum: Number: Minimum allowed slaving input.

maximum: Number: Maximum allowed slaving input.

Response:

OK

Context: String: Axis

Action: String: SETINPUTANALOGVELOCITY

Errors

TBD

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED

Example:

SET INPUT ANALOGVELOCITY PAN 0 MM -10 10¶

OK{PAN:SETINPUTANALOGVELOCITY}¶

EVT{42:AXISMODIFIED}

OK{PAN:AXISMODIFIED}

SET INPUT ENCODER

The SET INPUT ENCODER command configures joystick control for an axis in position mode. Joystick control input sent to the Furio head for an axis configured in this mode translates to a position. The position of the axis is calculated based on the difference between consecutive joystick input commands. The ratio is used as a multiplier when calculating the position.

NOTE: Command SET INPUT ENCODER is not supported in Furio Firmware. Use SET INPUT ANALOGVELOCITY instead.

Command:

SET INPUT ENCODER axis initial_value GR ratio¶

Arguments:

axis: String: The axis to configure.

initial_value: Number: The initial encoder value.

GR: Constant: GR stands for Gear Ratio

ratio: Number: The ratio to multiply with the encoder value to calculate the position.

Response:

OK

```
      Context: String: Axis
      Action: String: SETINPUTENCODER
```

Errors

TBD

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED

Example:

SET INPUT ENCODER FOCUS 0 GR 32¶

OK{FOCUS:SETINPUTENCODER}¶

EVT{42:AXISMODIFIED}
OK{PAN:AXISMODIFIED}

SET LOW LIMIT

The SET LOW LIMIT command is used to set and unset a temporary or persistent position limit for the 'lower' end of an axis.

- To set a temporary low limit, use Type=TEMPORARY. The command will toggle the low limit between the persistent limit and the current position.
- To set a persistent low limit, use Type=PERSISTENT. The command will toggle the low persistent limit between the low limit configured in the template and the current position. Setting persistent limits through the Furio API is not supported for the XY axis.

The active low limit can be verified with the AXIS STATUS command.

Temporary limits are lost when the robot is rebooted.

Command:

SET LOWLIMIT axis [type]¶

Arguments:

```
axis: String: The axis to configure.
Type: String; Optional; value can be "PERSISTENT" or "TEMPORARY"
      If Type param is missed, it is for set temporary limit.
      XY does not support <type> param
```

Response:

```
OK
  Context: String: *
```

Action: String: SETLOWLIMIT

Errors

LIMITEXCLUDESHOME

Synchronous Status Events:

None

Asynchronous Status Events:

LIMITSET/LIMITCLEARED

If Low temporary limit is set, LIMITSET event will be sent to status port;

If Low temporary limit is unset, LIMITCLEARED event will be sent to status port;

High Persistent limit set/unset, does not trigger a status event.

Example:

SET LOWLIMIT PAN¶

OK{PAN:SETLOWIMIT}¶

SET RATIO

The SET RATIO command configures the maximum speed achievable when direct control demand is at its maximum.

Command:

SET RATIO axis ratio¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

ratio: Number: percentage value between 0 and 100

Response:

OK

Context: String: axis

Action: String: SET RATIO

Errors

NOTAXIS

BADRATIOARG

Synchronous Status Events:

None

Asynchronous Status Events:

AXISMODIFIED – if the axis ratio value changes

Example:

SET RATIO PAN 100¶

OK{PAN:SETRATIO}¶

EVT{42:AXISMODIFIED}
OK{PAN:AXISMODIFIED}

SET DYNAMICLIMITSOVERRIDE

The SET DYNAMICLIMITSOVERRIDE command toggles dynamic limit override with axis's current position.

Command:

SET DYNAMICLIMITSOVERRIDE axis override¶

Arguments:

axis: String: one of the axis names returned by AXIS LIST

override: Number: 1 0r 0

Response:

OK

Context: String: axis

Action: String: SET RATIO

Errors

NOTAXIS

ARGTYPE

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

SET DYNAMICLIMITSOVERRIDE PAN 1¶

OK{: DYNAMICLIMITSOVERRIDE}¶*

SLAVING DISABLE

The SLAVING DISABLE command releases joystick control. Invoked without argument, SLAVING DISABLE removes joystick control from whichever session has control. The session_id argument allows a session to release direct control by supplying its session id (see command SESSION).

Command:

SLAVING DISABLE [session_id]¶

Arguments:

session_id (optional): String: a session ID

Response:

OK

Context: String: *

Action: String: SLAVINGDISABLE

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

SLAVINGENABLE, with the EVT record's context set to'*' to signify that no session has control.

Example:

Command Channel	Event Channel
SESSION¶	
OK{42:SESSION}¶	
SLAVING DISABLE¶	
OK{:SLAVINGDISABLE}¶*	
	EVT{*:SLAVINGENABLE}¶
	OK{*:SLAVINGENABLE}¶

SLAVING ENABLE

The SLAVING ENABLE command enables joystick control for this session. Only one session can have joystick control at the same time. Joystick input can arrive over either TCP (see SLAVING SET INPUTS below) or UDP channels.

The optional FORCE argument is historical; both SLAVING ENABLE and SLAVING ENABLE FORCE commands immediately take joystick control.

SLAVING ENABLE REQUEST will take joystick control only if robot is not controlled by another session; otherwise, it returns the error "INUSE".

Command:

SLAVING ENABLE [FORCE | REQUEST]¶

Arguments:

FORCE or REQUEST: String

Response:

OK

Context: String: *

Action: String: SLAVINGENABLE

Errors:

INUSE - response to a SLAVING ENABLE REQUEST if another session already has control. In this case, the context will be the id of the session with control.

Synchronous Status Events:

SLAVINGENABLE, with the EVT record's context set to the session id of the session granted control.

An INFO record indicates the controlling session's id and name, if the robot supports it (See command SYSTEM INFO, response argument SLAVINGREQUEST),

Asynchronous Status Events:

None

Example:

Command Channel	Event Channel
SESSION¶	
OK{42:SESSION}¶	
SLAVING ENABLE¶	
OK{:SLAVINGENABLE}¶*	
	EVT{42:SLAVINGENABLE}¶
	INFO{42:NAME=Station1}¶
	OK{:SLAVINGENABLE}¶*

Requesting control from another session:

SLAVING ENABLE REQUEST¶

ERR{42:SLAVINGENABLE=INUSE}

SLAVING SET INPUTS

The SLAVING SET INPUTS command is used to set joystick control input values for one or more axes. The values can be either velocities or position offsets, depending on the configuration of the respective axis.

Command:

SLAVING SET INPUTS axis1 value1 axis2 value2 …¶

Arguments:

axis: String: The axis to configure.

value: Number/Float: The new control value for that axis. For pan, tilt, zoom, lift, and track this is usually an integer representing a velocity. The range in these cases is [-something, +something] with the sign determining the direction of motion. Focus is usually configured as an absolute position.

For pedestal-based systems (see also **CamBot Specific Commands on page 100)**:

1) in VECTOR steering mode,
the XY axis encodes two velocities, one for the X axis in the integral portion of the value, and one for the Y axis in the fractional portion of the value. Negative velocities for Y are possible by the convention of adding a fixed offset to the Y value, for example:

- vel_x= 0, vel_y= 0 encoded as SLAVING SET INPUTS XY 0.128
- vel_x= 128, vel_y= 128 encoded as SLAVING SET INPUTS XY 128.256
- vel_x= -128, vel_y= -128 encoded as SLAVING SET INPUTS XY -128.0

2) in ROTATE steering mode,
the XY axis encodes a single velocity as an integral number, for example:

SLAVING SET INPUTS XY 58

3) in TRACK steering mode,
the XY axis encodes two velocities, one for the left wheel in the integral portion of the value, and one for the Y axis in the fractional portion of the value. Negative velocities for Y are possible by the convention of adding a fixed offset to the Y value as with VECTOR steering mode.

The following examples assume the input range was defined as [128, 127] by the SET INPUT ANALOGVELOCITY XY … command, and that a step speed of "50" is desired:

To move forward:
SLAVING SET INPUTS XY 50.178

To stop:
SLAVING SET INPUTS XY 0.128

To move backwards:
SLAVING SET INPUTS XY -50.78

To rotate in place clockwise:
SLAVING SET INPUTS XY 50.78

To rotate in place counter-clockwise:
SLAVING SET INPUTS XY -50.178

4) JOG steering mode works the same as TRACK steering mode, except that pan compensation will be disabled for Artimo so that rotating the base will rotate the whole robot including the pan stage. This allows a more intuitive pan relative steering when jogging Artimo. Added in 7.2d for Artimo.

Response:

OK

Context: String: *

Action: String: SLAVINGSETINPUTS

Errors

NOTMASTER

NOTINIT

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

SLAVING SET INPUTS PAN 10 TILT 25¶

OK{PAN:SLAVINGSETINPUTS}¶

Category Commands

CATEGORY ADD

The CATEGORY ADD command creates a new Category. The Furio head responds back with the generated Category ID.

Command:

CATEGORY ADD caption¶

Arguments:

caption: String:

Response:

OK

Context: String: Category ID

Action: String: CATEGORYADD

Errors

NOCAPTIONARG (Furio) / NOARG (Robotics Server)

UNEXPECTED (Furio)

Synchronous Status Events:

CATEGORYADD

Asynchronous Status Events:

None

Example:

CATEGORY ADD Test\.Category¶

OK{5:CATEGORYADD}¶

CATEGORY CLONE

The CATEGORY CLONE command creates a new Category identical to the source Category. The Furio head responds back with the generated Category ID.

Command:

CATEGORY CLONE category_id caption [suffix]¶

Arguments:

category_id: Number: The Category ID of the Category to clone.

caption: String: The name of the new category

suffix: String: an optional suffix to append to each cloned Preset. (Added in Furio 6.0)

Response:

OK

Context: String: Category ID of the new Category

Action: String: CATEGORYCLONE

Errors

NOCATEGORYIDARG (Furio)/ NOARG (Robotics Server)

NOCAPTIONARG (Robotics Server) CATEGORYNOTFOUND

UNEXPECTED (Furio)

Synchronous Status Events:

CATEGORYADD

Asynchronous Status Events:

None

Examples:

CATEGORY CLONE 100 NewCategory¶

OK{105:CATEGORYCLONE}¶

CATEGORY DELETE

The CATEGORY DELETE command deletes an existing Category. A Category can only be deleted when it is empty, unless the FORCE optional argument is specified. Categories containing locked items cannot be deleted.

Command:

CATEGORY DELETE category_id [FORCE]¶

Arguments:

category_id: Number

Response:

OK

Context: String: Category ID

Action: String: CATEGORYDELETE

Errors

NOCATEGORYIDARG (Furio) / NOARG (Robotics Server)

BADCATEGORYIDARG (Furio only, Robotics Server returns CATEGORYNOTFOUND)

NODEFAULTCATEGORYDELETE

CATEGORYNOTFOUND

NOTEMPTY

CATEGORYLOCKED

PRESETLOCKED

Synchronous Status Events:

CATEGORYDELETE

Asynchronous Status Events:

None

Example:

CATEGORY DELETE 3¶

OK{3:CATEGORYDELETE}¶

CATEGORY INFO

The CATEGORY INFO command returns details of the specified Category.

Command:

CATEGORY INFO category_id¶

Arguments:

category_id: Number

Response:

```
INFO
   Context: Category ID
   Name/Value Pairs:
         CAPTION: String:
OK
   Context: String: Category ID
   Action: String: CATEGORYINFO
```

Errors

NOCATEGORYIDARG (Furio) / NOARG (Robotics Server)

BADCATEGORYIDARG (Furio only, Robotics Server returns CATEGORYNOTFOUND)

CATEGORYNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

CATEGORY INFO 3¶

INFO{3:CAPTION=News\.Show}¶

OK{3:CATEGORYINFO}¶

CATEGORY LIST

The CATEGORY LIST command returns a list of all Categories.

Command:

CATEGORY LIST¶

Arguments:

None

Response:

```
INFO
   Context: Category ID
   Name/Value Pairs:
         CAPTION: String:
OK
   Context: String: *
   Action: String: CATEGORYLIST
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

```
CATEGORY LIST¶
INFO{3:CAPTION=General\.Use}¶
INFO{5:CAPTION=News\.Show}¶
INFO{9:CAPTION=Automation}¶
OK{*:CATEGORYLIST}¶
```

CATEGORY LOCK

The CATEGORY LOCK command locks all Presets and Moves in the specified Category. The category itself is not locked.

Command:

```
CATEGORY LOCK category_id¶
```

Arguments:

```
category_id: Number: The category to lock
```

Response:

```
OK
   Context: Number: category_id
   Action: String: CATEGORYLOCK
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

CATEGORY LOCK 4¶

OK{4:CATEGORYLOCK}¶

CATEGORY SET CAPTION

The CATEGORY SET CAPTION command changes the caption of an existing Category.

Command:

CATEGORY SET CAPTION category_id caption¶

Arguments:

category_id: Number:

caption: Number:

Response:

OK

Context: String: Category ID

Action: String: CATEGORYSETCAPTION

Errors

NOCATEGORYIDARG (Furio) / NOARG (Robotics Server)

BADCATEGORYIDARG (Furio only, Robotics Server returns CATEGORYNOTFOUND)

NODEFAULTCATEGORYSETCAPTION

CATEGORYNOTFOUND

NOCAPTIONARG

Synchronous Status Events:

CATEGORYMODIFIED (Furio only, legacy)

CATEGORYSETCAPTION

Asynchronous Status Events:

None

Example:

CATEGORY SET CAPTION 3 New\.Name¶

OK{3:CATEGORYSETCAPTION}¶

CATEGORY UNLOCK

The CATEGORY UNLOCK command unlocks all Presets and Moves in the specified Category. The category itself is never locked in the first place.

Command:

CATEGORY UNLOCK category_id¶

Arguments:

category_id: Number: The category to unlock

Response:

OK

Context: Number: category_id

Action: String: CATEGORYUNLOCK

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

CATEGORY UNLOCK 4¶

OK{4:CATEGORYUNLOCK}¶

Preset Commands

PRESET ADD

The PRESET ADD command stores the current position of the Furio head to a new Preset. The Furio head responds back with the generated Preset ID. When no Category ID is specified the Preset will be stored in the default category (0).

Command:

PRESET ADD caption [category_id]¶

Arguments:

caption: String: The name of the Preset. Data in this argument must be escaped.

category_id (optional): Number: The Category ID of the Category this Preset should be stored in.

Response:

OK

Context: Number: The Preset ID generated for this new Preset.

Action: String: PRESETADD

Errors

NOCAPTIONARG: No caption argument is given.

BADCATEGORYARG: The category_id argument is invalid (e.g. contained characters outside the range [0;9].

CATEGORYNOTFOUND: The category_id does not correspond to an existing Category.

UNEXPECTED: An error occurred when storing the Preset position.

Synchronous Status Events:

PRESETADD

Asynchronous Status Events:

None

Example:

PRESET ADD Test 345¶

OK{12346:PRESETADD}¶

PRESET CLEAR EXTERNAL ID

The PRESET CLEAR EXTERNAL ID command clears the External ID of a Preset. External IDs are mainly used with the RCCP Server.

Command:

PRESET CLEAR EXTERNALID preset_id ¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

Response:

OK

Context: String: Preset ID

Action: String: PRESETCLEAREXTERNALID

Errors

NOPRESETIDARG

PRESETNOTFOUND

Synchronous Status Events:

PRESETCLEAREXTERNALID

Asynchronous Status Events:

None

Example:

PRESET CLEAR EXTERNALID 100¶

OK{100:PRESETCLEAREXTERNALID}¶

PRESET CLONE

The PRESET CLONE command creates a new Preset identical to the source Preset.

Command:

PRESET CLONE preset_id [category_id]¶

Arguments:

preset_id: Number: The Preset ID of the Preset to clone.

category_id (optional): Number:

Response:

OK

Context: String: Preset ID of the new Preset

Action: String: PRESETCLONE

Errors

NOPRESETIDARG

PRESETNOTFOUND

BADCATEGORYARG

CATEGORYNOTFOUND

UNEXPECTED

Synchronous Status Events:

PRESETCLONE

Asynchronous Status Events:

None

Examples:

PRESET CLONE 100¶

OK{113:PRESETCLONE}¶

PRESET COPY

The PRESET COPY command copies all properties from one Preset to another. This includes position, acceleration, deceleration, duration, etc.

Command:

PRESET COPY source_preset_id target_preset_id¶

Arguments:

source_preset_id: Number: The Preset ID of the Preset to use as source of the copy.

target_preset_id: Number: The Preset ID of the Preset to use as destination of the copy.

Response:

OK

Context: String: Target Preset ID

Action: String: PRESETCOPY

Errors

NOPRESETIDARG

NOTARGETIDARG

PRESETNOTFOUND

PRESETLOCKED

Synchronous Status Events:

PRESETMODIFIED

Asynchronous Status Events:

None

Examples:

PRESET COPY 100 101¶

OK{101:PRESETCOPY}¶

PRESET CUE

The PRESET CUE command prepares the Furio head to move to the position of the specified Preset as quickly as possible. This command is used to align the XY axis of CamBot pedestals to eliminate recall delays associated with pre-recall rotation.

Command:

PRESET CUE preset_id [ALTVELOCITY]¶

Arguments:

preset_id: Number: The Preset ID of the Preset to cue to.

ALTVELOCITY (optional): Constant: Use pre-configured alternate velocity limit when executing the motion.

Response:

OK

Context: String: Preset ID

Action: String: PRESETCUE

Errors

NOPRESETIDARG

COMMAND – if sent to a head that does not have an XY axis

BEYONDLIMITS

Synchronous Status Events:

PRESETCUE

Asynchronous Status Events:

PRESETCUEFINISHED

Examples:

PRESET CUE 100¶

OK{100:PRESETCUE}¶

PRESET CUT

The PRESET CUT command instructs the Furio head to move to the position of the specified Preset as quickly as possible. Each axis moves to the target position unsynchronized using the maximum acceleration, speed, and deceleration.

Command:

PRESET CUT preset_id [ALTVELOCITY]¶

Arguments:

preset_id: Number: The Preset ID of the Preset to cut to.

ALTVELOCITY (optional): Constant: Use pre-configured alternate velocity limit when executing the motion.

Response:

OK

Context: String: Preset ID

Action: String: PRESETCUT

Errors

NOPRESETIDARG

BEYONDLIMITS

Synchronous Status Events:

PRESETCUT
PRESETCUTREQUESTED

Asynchronous Status Events:

PRESETCUTFINISHED

Examples:

PRESET CUT 100¶

OK{100:PRESETCUT}¶

PRESET DELETE

The PRESET DELETE command deletes a Preset. Preset cannot be deleted when they are locked using the PRESET LOCK command or used in a Move.

Command:

PRESET DELETE preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to delete.

Response:

OK

Context: String: Preset ID

Action: String: PRESETDELETE

Errors

NOPRESETIDARG

PRESETNOTFOUND

PRESETLOCKED

PRESETINUSE

Synchronous Status Events:

PRESETDELETE

Asynchronous Status Events:

None

Example:

PRESET DELETE 100¶

OK{100:PRESETDELETE}¶

PRESET GOTO

The PRESET GOTO command instructs the Furio head to move to the position of the specified Preset over a certain time period. If no time is specified, the Duration property of the Preset will be used as duration of the movement. Each axis uses the acceleration and deceleration value specified in the Preset. All axes move to the target position in a synchronized way.

Command:

PRESET GOTO preset_id [duration]¶

Arguments:

preset_id: Number: The Preset ID of the Preset to execute.

duration (optional): Float: alternate recall duration, in seconds.

Response:

OK

Context: String: Preset ID

Action: String: PRESETGOTO

Errors

NOPRESETIDARG

PRESETNOTFOUND

BADDURATIONARG

INFO: MINIMUMDURATION: Number:

BEYONDLIMITS

Synchronous Status Events:

PRESETGOTO

Asynchronous Status Events:

PRESETGOTOFINISHED

Example:

PRESET GOTO 100¶

OK{100:PRESETGOTO}¶

PRESET INFO

The PRESET INFO command returns details of the specified Preset.

Command:

PRESET INFO preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to list.

Response:

INFO

Context: Number: Preset ID

Name/Value Pairs:

CAPTION: String:
CATEGORY: Number: a category ID
DURATION: Float: default GOTO duration in seconds
LOCKED: Boolean:
AUTOROTATE: Boolean (XY peds only)
EXTERNALID: Number:

INFO (1 per axis)
Context: Number: Preset ID
Name/Value Pairs:
AXIS: String:
MOTORPOSITION: Number: axis position in encoder counts
POSITION: Float: axis position in real world units – see note in example below about XY position representation.
ACCELERATION: Number
DECELERATION: Number

OK
Context: Number: Preset ID
Action: String: PRESETINFO

Errors

NOPRESETIDARG
PRESETNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Examples:

PRESET INFO 2

INFO{2:CAPTION=test,CATEGORY=1,DURATION=100.000000,EXTERNALID=3}¶

INFO{2:AXIS=ZOOM,MOTORPOSITION=470,POSITION=470.000000,ACCELERATION=25,DECELERATION=25}¶

INFO{2:AXIS=FOCUS,MOTORPOSITION=50641,POSITION=50641.000000,ACCELERATION=25,DECELERATION=25}¶

INFO{2:AXIS=PAN,MOTORPOSITION=124555,POSITION=51.257613,ACCELERATION=100,DECELERATION=25}¶

INFO{2:AXIS=TILT,MOTORPOSITION=-1938,POSITION=-0.797531,ACCELERATION=25,DECELERATION=25}¶

INFO{2:AXIS=LIFT,MOTORPOSITION=2541379,POSITION=-1931.638113,ACCELERATION=25,DECELERATION=25}¶

INFO{2:AXIS=TRACK,MOTORPOSITION=97573,POSITION=863.480074,ACCELERATION=25,DECELERATION=25}¶

OK{2:PRESETINFO}¶

For XY axis, X, Y and THETA positions are combined into a single floating point value. See AXIS STATUS description of MTRPOS for encoding details

In the example below, this would result in Theta=134, X=-3504, Y=4688

INFO{2:AXIS=XY,POSITION=134057456.065648,ACCELERATION=25,DECELERATION=25}

PRESET LIST

The PRESET LIST command returns a list of all Presets. Optionally a Category ID can be supplied. In that case only the Presets in the specified Category will be listed.

Command:

PRESET LIST [category_id]¶

Arguments:

category_id (optional): Number: The Category ID of the Presets to list.

Response:

INFO

Context: Number: Preset ID

Name/Value Pairs:

CAPTION: String

CATEGORY: Number: a category ID

DURATION: Float: default GOTO duration in seconds

LOCKED: Boolean

AUTOROTATE: Boolean (XY peds only)

EXTERNALID: Number

OK

Context: String: *

Action: String: PRESETLIST

Errors

BADCATEGORYIDARG

CATEGORYNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

PRESET LIST¶

INFO{100:CAPTION=Some\.Preset,DURATION=100.000000}¶

INFO{101:CAPTION=Another\.Preset,CATEGORY=1,DURATION=10.000000}¶

INFO{110:CAPTION=Last\.Preset,CATEGORY=5,DURATION=100.000000}¶

OK{:PRESETLIST}¶*

PRESET LOCK

The PRESET LOCK command locks a Preset. Presets that are locked cannot be deleted or edited in any way.

Command:

PRESET LOCK preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to lock.

Response:

OK

Context: String: Preset ID

Action: String: PRESETLOCK

Errors

NOPRESETIDARG

PRESETNOTFOUND

Synchronous Status Events:

PRESETLOCK

Asynchronous Status Events:

None

Example:

PRESET LOCK 100¶

OK{100:PRESETLOCK}¶

PRESET SET ACCELERATION

The PRESET SET ACCELERATION command changes the acceleration value of an Axis of a Preset. This value is used when executing the Preset Goto command.

Command:

PRESET SET ACCELERATION preset_id axis value¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

axis: String:

value: Number: value between 1 and 100

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETACCELERATION

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOAXISARG

BADAXISARG

AXISNOTFOUND

NOVALUEARG

BADVALUEARG

PRESETLOCKED

Synchronous Status Events:

PRESETSETACCELERATION

Asynchronous Status Events:

None

Example:

PRESET SET ACCELERATION 100 PAN 50¶

OK{100:PRESETSETACCELERATION}¶

PRESET SET CAPTION

The PRESET SET CAPTION command changes the caption of an existing Preset.

Command:

PRESET SET CAPTION preset_id caption¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

caption: String: The new name of the Preset. Data in this argument must be escaped.

Response:

OK

Context: Number: The Preset ID of the Preset whose caption is changed.

Action: String: PRESETSETCAPTION

Errors

NOPRESETIDARG: No preset_id argument is given.

PRESETNOTFOUND: The preset_id does not correspond to an existing Preset.

NOCAPTIONARG: No caption argument is given.

PRESETLOCKED

Synchronous Status Events:

PRESETSETCAPTION

Asynchronous Status Events:

None

Example 1:

PRESET SET CAPTION 12346 Test¶

OK{12346:PRESETSETCAPTION}¶

Example 2:

PRESET SET CAPTION 12347 Wide\.shot\.1¶

OK{12347:PRESETSETCAPTION}¶

PRESET SET CATEGORY

The PRESET SET CATEGORY command moves a Preset to the specified Category.

Command:

PRESET SET CATEGORY preset_id category_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

category_id: Number:

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETCATEGORY

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOCATEGORYARG

BADCATEGORYARG

CATEGORYNOTFOUND

PRESETLOCKED

Synchronous Status Events:

PRESETSETCATEGORY

Asynchronous Status Events:

None

Example:

PRESET SET CATEGORY 100 10¶

OK{100:PRESETSETCATEGORY}¶

PRESET SET DECELERATION

The PRESET SET DECELERATION command changes the deceleration value of an Axis of a Preset. This value is used when executing the Preset Goto command.

Command:

PRESET SET DECELERATION preset_id axis value¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

axis: String:

value: Number: value between 1 and 100

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETDECELERATION

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOAXISARG

BADAXISARG

AXISNOTFOUND

NOVALUEARG

BADVALUEARG

PRESETLOCKED

Synchronous Status Events:

PRESETSETDECELERATION

Asynchronous Status Events:

None

Example:

PRESET SET DECELERATION 100 PAN 50¶

OK{100:PRESETSETDECELERATION}¶

PRESET SET DURATION

The PRESET SET DURATION command changed the duration of a Preset that is used when executing the PRESET GOTO command.

Command:

PRESET SET DURATION preset_id duration¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

duration: Float: the new default GOTO duration, in seconds.

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETDURATION

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOVALUEARG

INVALIDNUMBER

PRESETLOCKED

Synchronous Status Events:

PRESETSETDURATION

Asynchronous Status Events:

None

Example:

PRESET SET DURATION 100 10¶

OK{100:PRESETSETDURATION}¶

PRESET SET EXTERNAL ID

The PRESET SET EXTERNAL ID command changes the External ID of a Preset. External IDs are mainly used with the RCCP Server.

Command:

PRESET SET EXTERNALID preset_id external_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

external_id: Number:

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETEXTERNALID

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOVALUEARG

INVALIDNUMBER

PRESETLOCKED

Synchronous Status Events:

PRESETSETEXTERNALID

Asynchronous Status Events:

None

Example:

PRESET SET EXTERNALID 100 10¶

OK{100:PRESETSETEXTERNALID}¶

PRESET SET POSITION

The PRESET SET POSITION command overwrites the position of an existing Preset with the current position.

Command:

PRESET SET POSITION preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETPOSITION

Errors

NOPRESETIDARG

PRESETNOTFOUND

PRESETLOCKED

Synchronous Status Events:

PRESETSETPOSITION

Asynchronous Status Events:

None

Example:

PRESET SET POSITION 100¶

OK{100:PRESETSETPOSITION}¶

PRESET SET AUTOROTATE

The PRESET SET AUTOROTATE command sets an existing Preset's auto rotate to be true or false. When the auto rotate is true, Preset will rotate to the defined angle after the Preset recall is over. This command is only used on CambotXY system.

Command:

PRESET SET AUTOROTATE preset_id enable¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

enable: string: "true" or "false".

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETAUTOROTATE

Errors

NOPRESETIDARG

PRESETNOTFOUND

NOENABLEARG

PRESETLOCKED

BADENABLEARG

NOPEDAXIS

Synchronous Status Events:

PRESETSETPOSITION

Asynchronous Status Events:

None

Example:

PRESET SET AUTOROTATE 100 1¶

OK{100:PRESETSETAUTOROTATE}¶

PRESET MINIMUMDURATION

The PRESET MINIMUMDURATION command gets the preset recall minimum required time.

Command:

PRESET MINIMUMDURATION preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

Response:

OK

Context: String: Preset ID

Action: String: PRESETMINIMUMDURATION

Errors

NOPRESETIDARG

PRESETNOTFOUND

Synchronous Status Events:

PRESETSETPOSITION

Asynchronous Status Events:

None

Example:

PRESET PRESETMINIMUMDURATION 100¶

INFO{100:MINIMUMDURATION=2.60}

OK{100: PRESETMINIMUMDURATION}¶

PRESET UNLOCK

The PRESET UNLOCK command unlocks a Preset. Presets that are unlocked can be deleted.

Command:

PRESET UNLOCK preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset to unlock.

Response:

OK

Context: String: Preset ID

Action: String: PRESETLOCK

Errors

NOPRESETIDARG

PRESETNOTFOUND

Synchronous Status Events:

PRESETUNLOCK

Asynchronous Status Events:

None

Examples:

PRESET UNLOCK¶

OK{100:PRESETUNLOCK}¶

PRESET USAGE

The PRESET USAGE command lists the Moves in which the specified Preset is being used.

Command:

PRESET USAGE preset_id¶

Arguments:

preset_id: Number: The Preset ID of the Preset whose usage should be listed.

Response:

```
INFO
   Context: Move ID
   Name/Value Pairs:
         DISTANCE: Number:
OK
   Context: String: Preset ID
   Action: String: PRESETUSAGE
```

Errors

```
NOPRESETIDARG
PRESETNOTFOUND
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
None
```

Example:

```
PRESET USAGE 100¶
INFO{20:DISTANCE=0}¶
INFO{45:DISTANCE=10}¶
OK{100:PRESETUSAGE}¶
```

QUICKFOCUS BEGIN

The QUICKFOCUS BEGIN command starts a quick focus process if axes are idle.

Command:

```
QUICKFOCUS BEGIN¶
```

Arguments:

```
None.
```

Response:

```
OK
      Context: String: *
      Action: String: QUICKFOCUSBEGIN
```

Errors

```
NOTAXIS
QUICKFOCUSALREADYINPROGRESS
FAIL
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

None

Example:

QUICKFOCUS BEGIN¶

OK{:QUICKFOCUSBEGIN}¶*

QUICKFOCUS END

The QUICKFOCUS END command finishes quick focus operation.

Command:

QUICKFOCUS END¶

Arguments:

None.

Response:

OK

Context: String: *

Action: String: QUICKFOCUSEND

Errors

NOTAXIS

QUICKFOCUSNOTSTARTED

FAIL

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

QUICKFOCUS END¶

OK{:QUICKFOCUSEND}¶*

Move Commands

MOVE ADD

The MOVE ADD command created a new (empty) move. The Furio head responds back with the generated Move ID. When no Category ID is specified the Move will be stored in the default category (0).

Command:

MOVE ADD caption [category_id]¶

Arguments:

caption: String

category_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVEADD

Errors

NOCAPTIONARG

CATEGORYNOTFOUND

Synchronous Status Events:

MOVEADD

Asynchronous Status Events:

None

Example:

MOVE ADD Test\.Move 1¶

OK{10:MOVEADD}¶

MOVE KEYFRAME ADD

The MOVE KEYFRAME ADD command adds a Keyframe to a Move. A Preset and distance is specified to create the Keyframe. The Move will reach to the position of the Preset after the defined time (if within the physical limits of the system).

Command:

MOVE KEYFRAME ADD move_id preset_id distance¶

Arguments:

move_id: Number:

preset_id: Number:

distance: Float: position on the move timeline, in seconds

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMEADD

Errors

NOMOVEIDARG

NOPRESETIDARG

BADPRESETIDARG

PRESETNOTFOUND

NODISTANCEARG

BADDISTANCEARG

MOVENOTFOUND

DUPLICATEDISTANCE

Synchronous Status Events:

MOVEKEYFRAMEADD

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME ADD 10 3 15¶

OK{10:MOVEKEYFRAMEADD}¶

MOVE KEYFRAME ADD (while running move)

The MOVE KEYFRAME ADD command adds a dynamically generated Keyframe to a running Move. A new Preset is generated automatically at the current position of the Furio when the command is received and placed in the Default category. A new Keyframe is added to the move at the time distance equal to the running move's current execution time at which the command is received. The request is validated upon reception, but the move is modified only once the move finishes running.

Command:

MOVE KEYFRAME ADD move_id¶

Arguments:

move_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMEADD

Errors

NOMOVEIDARG

NOPRESETIDARG
BADPRESETIDARG
PRESETNOTFOUND
NODISTANCEARG
BADDISTANCEARG
MOVENOTFOUND
DUPLICATEDISTANCE

Synchronous Status Events:

MOVEKEYFRAMEADD

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME ADD 10¶

OK{10:MOVEKEYFRAMEADD}¶

MOVE KEYFRAME AXIS DISABLE

The MOVE KEYFRAME AXIS DISABLE command disabled an Axis of a Keyframe of a Move. When an Axis of a Keyframe is disabled the position of that Axis in the reference Preset is not taken into account when executing the Move. Other (enabled) Axes of that Keyframe are still taken into account.

Command:

MOVE KEYFRAME AXIS DISABLE move_id distance axis¶

Arguments:

move_id: Number:
distance: Float: position on the move timeline, in seconds
axis: Number:

Response:

OK
  Context: String: Move ID
  Action: String: MOVEKEYFRAMEAXISDISABLE

Errors

NOMOVEIDARG
NODISTANCEARG
NOAXISARG
BADAXISARG
MOVENOTFOUND
NOKEYATDISTANCE

Synchronous Status Events:

MOVEKEYFRAMEAXISDISABLE

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME AXIS DISABLE 3 15 PAN¶

OK{3:MOVEKEYFRAMEAXISDISABLE}¶

MOVE KEYFRAME AXIS ENABLE

The MOVE KEYFRAME AXIS ENABLE command enables an Axis of a Keyframe of a Move. When an Axis of a Keyframe is enabled the position of that Axis in the reference Preset is taken into account when executing the Move.

Command:

MOVE KEYFRAME AXIS ENABLE move_id distance axis¶

Arguments:

move_id: Number:

distance: Float: position on the move timeline, in seconds axis: String:

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMEAXISENABLE

Errors

NOMOVEIDARG

NODISTANCEARG

NOAXISARG

BADAXISARG

MOVENOTFOUND

NOKEYATDISTANCE

Synchronous Status Events:

MOVEKEYFRAMEAXISENABLED

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME AXIS ENABLE 3 15 PAN¶

OK{3:MOVEKEYFRAMEAXISENABLE}¶

MOVE KEYFRAME DELETE

The MOVE KEYFRAME DELETE command deletes a Keyframe from a Move. Keyframes can only be deleted when a Move is not locked.

Command:

MOVE KEYFRAME DELETE move_id distance¶

Arguments:

move_id: Number

distance: Float: position on the move timeline, in seconds

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMEDELETE

Errors

NOMOVEIDARG

NODISTANCEARG MOVEKEYFRAMEADD

MOVENOTFOUND

NOKEYATDISTANCE

Synchronous Status Events:

MOVEKEYFRAMEDELETE

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME DELETE 10 15¶

OK{10:MOVEKEYFRAMEDELETE}¶

MOVE KEYFRAME INFO

The MOVE KEYFRAME INFO command returns details of the specified Keyframe in a Move.

Command:

MOVE KEYFRAME INFO move_id distance¶

Arguments:

move_id: Number:

distance: Float: position on the move timeline, in seconds

Response:

INFO

Context: Float: Distance

Name/Value Pairs:

PRESET: Number: preset ID

```
INFO (1 per axis)
   Context: Float: Distance
   Name/Value Pairs:
         AXIS: String:
         ENABLED: Boolean (TRUE / FALSE)
         RECORDING: String (NONE / ENABLED / DISABLED)
         TENSION: Number tension value (optional)
OK
   Context: Number: Move ID
   Action: String: MOVEKEYFRAMEINFO
```

Errors

```
NOMOVEIDARG
NODISTANCEARG
MOVENOTFOUND
NOKEYATDISTANCE
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
None
```

Examples:

```
MOVE KEYFRAME INFO 1 10
INFO{10.00000000000000000000000000:PRESET=4}
INFO{10.00000000000000000000000000:AXIS=XY,ENABLED=TRUE,RECORDING=NO
NE,TENSION=30}
INFO{10.00000000000000000000000000:AXIS=ZOOM,ENABLED=TRUE,RECORDING=
NONE}
INFO{10.00000000000000000000000000:AXIS=FOCUS,ENABLED=TRUE,RECORDING
=NONE,TEN
INFO{10.00000000000000000000000000:AXIS=PAN,ENABLED=TRUE,RECORDING=N
ONE}
INFO{10.00000000000000000000000000:AXIS=TILT,ENABLED=TRUE,RECORDING=
NONE}
INFO{10.00000000000000000000000000:AXIS=LIFT,ENABLED=TRUE,RECORDING=
NONE}
```

MOVE KEYFRAME SET ACCELERATION

The MOVE KEYFRAME SET ACCELERATION command changes the acceleration of an Axis of a Keyframe.

Command:

MOVE KEYFRAME SET ACCELERATION move_id distance axis value¶

Arguments:

move_id: Number:
distance: Float: position on the move timeline, in seconds
axis: String:
value: Number: a value between 1 and 100

Response:

OK
Context: String: Move ID
Action: String: MOVEKEYFRAMESETACCELERATION

Errors

NOMOVEIDARG
NODISTANCEARG
NOAXISARG
BADAXISARG
NOVALUEARG
MOVENOTFOUND
NOKEYATDISTANCE
BADACCELERATIONARG

Synchronous Status Events:

MOVEKEYFRAMESETACCELERATION

Asynchronous Status Events:

None

Examples:

MOVE KEYFRAME SET ACCELERATION 3 15 PAN 50¶

OK{3:MOVEKEYFRAMESETACCELERATION}¶

MOVE KEYFRAME SET DECELERATION

The MOVE KEYFRAME SET DECELERATION command changes the acceleration of an Axis of a Keyframe.

Command:

MOVE KEYFRAME SET DECELERATION move_id distance axis value¶

Arguments:

move_id: Number:
distance: Float: position on the move timeline, in seconds
axis: String:
value: Number: a value between 1 and 100

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMESETDECELERATION

Errors

NOMOVEIDARG

NODISTANCEARG

NOAXISARG

BADAXISARG

NOVALUEARG

MOVENOTFOUND

NOKEYATDISTANCE

BADDECELERATIONARG

Synchronous Status Events:

MOVEKEYFRAMESETDECELERATION

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME SET DECELERATION 3 15 PAN 50¶

OK{3:MOVEKEYFRAMESETDECELERATION}¶

MOVE KEYFRAME SET TENSION

The MOVE KEYFRAME SET TENSION command changes the tension of a Keyframe for an XY axis.

Introduced in Furio version 5.2.

Command:

MOVE KEYFRAME SET TENSION move_id distance axis value¶

Arguments:

move_id: Number:

distance: Float: position on the move timeline, in seconds

axis: String:

value: Number: a value between 0 and 100

Response:

OK

Context: String: Move ID

Action: String: MOVEKEYFRAMESETTENSION

Errors

NOMOVEIDARG

NODISTANCEARG
NOAXISARG
BADAXISARG
NOVALUEARG
MOVENOTFOUND
NOKEYATDISTANCE
BADTENSIONARG

Synchronous Status Events:

MOVEKEYFRAMESETTENSION

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME SET TENSION 3 15.0 XY 50¶

OK{3:MOVEKEYFRAMESETTENSION}¶

MOVE KEYFRAME SET DISTANCE

The MOVE KEYFRAME SET DISTANCE command changes the distance of a Keyframe in a Move. It thus shifts the Keyframe on the timeline of the Move.

Command:

MOVE KEYFRAME SET DISTANCE move_id distance new_distance¶

Arguments:

move_id: Number:
distance: Float: old position on the move timeline, in seconds
new_distance: Float: new position on the move timeline, in seconds

Response:

OK
  Context: String: Move ID
  Action: String: MOVEKEYFRAMESETDISTANCE

Errors

NOMOVEIDARG
MOVENOTFOUND
NODISTANCEARG
NOKEYATDISTANCE
NONEWDISTANCEARG
BADDISTANCEARG
DUPLICATEDISTANCE

Synchronous Status Events:

MOVEKEYFRAMESETDISTANCE

Asynchronous Status Events:

None

Example:

MOVE KEYFRAME SET DISTANCE 3 15 20¶

OK{3:MOVEKEYFRAMESETDISTANCE}¶

MOVE CLEAR EXTERNAL ID

The MOVE CLEAR EXTERNAL ID command clears the External ID of a Move. External IDs are mainly used with the RCCP Server.

Command:

MOVE CLEAR EXTERNALID move_id ¶

Arguments:

move_id: Number: The Move ID of the Move to alter.

Response:

OK

Context: String: Move ID

Action: String: MOVECLEAREXTERNALID

Errors

NOMOVEIDARG

MOVENOTFOUND

Synchronous Status Events:

MOVECLEAREXTERNALID

Asynchronous Status Events:

None

Example:

MOVE CLEAR EXTERNALID 100¶

OK{100:MOVECLEAREXTERNALID}¶

MOVE CLONE

The MOVE CLONE command creates a new Move identical to the source Move. All properties and keyframes are copied to the new Move. The Furio head replies with the Move ID of the clone. Optionally the new Move can be stored in a different category.

Command:

MOVE CLONE move_id [category_id]¶

Arguments:

move_id: Number:

category_id: Number:

Response:

OK

Context: String: Move ID of the new Move.

Action: String: MOVECLONE

Errors

NOMOVEIDARG

MOVENOTFOUND

BADCATEGORYID

CATEGORYNOTFOUND

UNEXPECTED

Synchronous Status Events:

MOVECLONE

Asynchronous Status Events:

None

Example:

MOVE CLONE 10¶

OK{15:MOVECLONE}¶

MOVE CUT

The MOVE CUT command instructs the Furio head to move to the position of the specified Move Keyframe as fast as possible. Each axis moves to the target position unsynchronized using the maximum acceleration, speed, and deceleration.

Command:

MOVE CUT move_id distance [ALTVELOCITY] axes ¶

Arguments:

move_id: Number:

distance: Float: position on the move timeline, in seconds

ALTVELOCITY (optional): Constant: Use pre-configured alternate velocity limit when executing the motion.

Axes: use this optional param for CUE RECORD; list of axes that will not move with the next move command (forward or record); I.e, these axes are recording axes

Response:

OK

Context: String: Move ID

Action: String: MOVE CUT

Errors

NOMOVEIDARG

MOVENOTFOUND

NOTIMEARG

BEYONDLIMITS

Synchronous Status Events:

MOVECUT

Asynchronous Status Events:

MOVECUTFINISHED

Example:

Regular Move CUE:

MOVE CUT 3 0¶

OK{3:MOVECUT}¶

For CUE RECORD:

MOVE CUT 1 0 PAN

MOVE DELETE

The MOVE DELETE command deletes a Move. Move cannot be deleted when they are locked using the MOVE LOCK command.

Command:

MOVE DELETE move_id¶

Arguments:

move_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVEDELETE

Errors

NOMOVEIDARG

MOVENOTFOUND

MOVELOCKED

Synchronous Status Events:

MOVEDELETE

Asynchronous Status Events:

None

Example:

MOVE DELETE 10¶

OK{10:MOVEDELETE}¶

MOVE FORWARD

The MOVE FORWARD command instructs the Furio head to start a Move. This Move needs to be previously prepared with MOVE CUT. Each axis uses the acceleration, deceleration and distance (time) values specified in the Move Keyframes. All axes move to the target positions in a synchronized way.

Command:

MOVE FORWARD move_id

Arguments:

move_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVEFORWARD

Errors

NOMOVEIDARG

MOVENOTFOUND

NOTIMEARG

BEYONDLIMITS

UNEXPECTED

Synchronous Status Events:

MOVEFORWARD

Asynchronous Status Events:

MOVEFORWARDFINISHED

Example:

MOVE FORWARD 10¶

OK{10:MOVEFORWARD}¶

MOVE INFO

The MOVE INFO command returns details of the specified Move.

Command:

MOVE INFO move_id¶

Arguments:

move_id: Number:

Response:

```
INFO
   Context: Number: Move ID
   Name/Value Pairs:
         CAPTION: String:
         CATEGORY: Number:
         DURATION: Float: default MOVE FORWARD duration, in seconds
         LOCKED: Boolean:
         EXTERNALID: Number:
         LOOP: Boolean:
         VALID: Boolean:
INFO (1 per keyframe)
   Context: Number: Preset ID
   Name/Value Pairs:
         PRESET: Number: Preset ID
INFO (1 per axis per keyframe)
   Context: Float: Keyframe Distance on the move timeline, in
seconds
   Name/Value Pairs:
         AXIS: String:
         ACCELERATION: Number
         DECELERATION: Number
         TENSION: Number (XY Axis only, new in Furio API 1.7)
         ENABLED: Boolean
OK
   Context: Number: Move ID
   Action: String: MOVEINFO
```

Errors

```
NOMOVEIDARG
MOVENOTFOUND
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
None
```

Example:

MOVE INFO 3¶

INFO{3:CAPTION=3,DURATION=10.000000}¶

INFO{0.000000:PRESET=2}¶

INFO{0.000000:AXIS=ZOOM,ENABLED=TRUE}¶

```
INFO{0.000000:AXIS=FOCUS,ENABLED=TRUE}¶
INFO{0.000000:AXIS=PAN,ACCELERATION=10,ENABLED=TRUE}¶
INFO{0.000000:AXIS=TILT,ENABLED=TRUE}¶
INFO{0.000000:AXIS=LIFT,ENABLED=TRUE}¶
INFO{0.000000:AXIS=TRACK,ENABLED=TRUE}¶
INFO{10.000000:PRESET=2}¶
INFO{10.000000:AXIS=ZOOM,ENABLED=TRUE}¶
INFO{10.000000:AXIS=FOCUS,ENABLED=TRUE}¶
INFO{10.000000:AXIS=PAN,ENABLED=TRUE}¶
INFO{10.000000:AXIS=TILT,ENABLED=TRUE}¶
INFO{10.000000:AXIS=LIFT,DECELERATION=70,ENABLED=TRUE}¶
INFO{10.000000:AXIS=TRACK,ENABLED=TRUE}¶
OK{3:MOVEINFO}¶
```

MOVE LIST

The MOVE LIST command returns a list of all Moves. Optionally a Category ID can be supplied. In that case only the Moves in the specified Category will be listed.

Command:

MOVE LIST [category_id]¶

Arguments:

```
category_id (optional): Number:
```

Response:

```
INFO
   Context: Number: Move ID
   Name/Value Pairs:
         CAPTION: String
         CATEGORY: Number
         DURATION: Float: default MOVE FORWARD duration, in seconds
         LOCKED: Boolean
         EXTERNALID: Number
OK
   Context: String: *
   Action: String: MOVELIST
```

Errors

```
BADCATEGORYIDARG
CATEGORYNOTFOUND
```

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE LIST¶

INFO{1:CAPTION=Move\.1,CATEGORY=1}¶

INFO{2:CAPTION=Move\.2,LOCKED=TRUE}¶

INFO{3:CAPTION=Move\.3,DURATION=10.000000}¶

OK{:MOVELIST}¶*

MOVE LOCK

The MOVE LOCK command locks a Move. Moves that are locked cannot be deleted.

Command:

MOVE LOCK move_id¶

Arguments:

move_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVELOCK

Errors

NOMOVEIDARG

MOVENOTFOUND

Synchronous Status Events:

MOVELOCK

Asynchronous Status Events:

None

Example:

MOVE LOCK 10¶

OK{10:MOVELOCK}¶

MOVE SET CAPTION

The MOVE SET CAPTION command changes the caption of an existing Move.

Command:

MOVE SET CAPTION move_id caption¶

Arguments:

move_id: Number:

caption: String:

Response:

OK

Context: String: Move ID

Action: String: MOVESETCAPTION

Errors

NOMOVEIDARG

NOCAPTIONIDARG

MOVENOTFOUND

Synchronous Status Events:

MOVESETCAPTION

Asynchronous Status Events:

None

Example:

MOVE SET CAPTION 11 New\.Caption¶

OK{1:MOVESETCAPTION}¶

MOVE SET CATEGORY

The MOVE SET CATEGORY command moves a Move to the specified Category.

Command:

MOVE SET CATEGORY move_id category_id¶

Arguments:

move_id: Number

category_id: Number

Response:

OK

Context: String: Move ID

Action: String: MOVESETCATEGORY

Errors

NOMOVEIDARG

MOVENOTFOUND

NOCATEGORYARG

BADCATEGORYARG

CATEGORYNOTFOUND

Synchronous Status Events:

MOVESETCATEGORY

Asynchronous Status Events:

None

Example:

MOVE SET CATEGORY 10 2¶

OK{10:MOVESETCATEGORY}¶

MOVE SET DURATION

The MOVE SET DURATION command changes the total duration of a Move, linearly scaling the intermediate keyframe distances.

Command:

MOVE SET DURATION move_id duration¶

Arguments:

move_id: Number

duration: Float: new default MOVE FORWARD duration, in seconds

Response:

OK

Context: String: Move ID

Action: String: MOVESETDURATION

Errors

NOARG

MOVENOTFOUND

BADDURATIONARG

Synchronous Status Events:

MOVESETDURATION

Asynchronous Status Events:

MOVEDURATIONMODIFIED

MOVEVALIDMODIFIED

Example:

MOVE SET DURATION 10 2¶

OK{10:MOVESETDURATION}¶

MOVE SET EXTERNAL ID

The MOVE SET EXTERNAL ID command changes the External ID of a Move. External IDs are mainly used with the RCCP Server.

Command:

MOVE SET EXTERNALID move_id external_id¶

Arguments:

move_id: Number: The Move ID of the Move to alter.

external_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVESETEXTERNALID

Errors

NOMOVEIDARG

MOVENOTFOUND

NOVALUEARG

INVALIDNUMBER

Synchronous Status Events:

MOVESETEXTERNALID

Asynchronous Status Events:

None

Example:

MOVE SET EXTERNALID 100 10¶

OK{100:MOVESETEXTERNALID}¶

MOVE SET LOOP

The MOVE SET LOOP command changes the Looped state of a Move. To be loopable, a Move's first and last keyframes must point to the same preset.

Command:

MOVE SET LOOP move_id enable ¶

Arguments:

move_id: Number: The Move ID of the Move to alter.

enable: Boolean: TRUE to enable looping, FALSE to disable

Response:

OK

Context: String: Move ID

Action: String: MOVESETLOOP

Errors

NOMOVEIDARG – The first argument, move_id, is missing

NOENABLEARG – The second argument, enable, is missing

BADENABLEARG – The second argument, enable, is not a boolean

MOVENOTFOUND – move_id references a move that does not exist

MOVELOCKED – move_id references a move that is locked

INVALIDPRESETSEQUENCE - Move cannot be looped. Either there are not enough
keyframes in the move or the first and last keyframes
do not reference the same preset.

Synchronous Status Events:

MOVESETLOOP

Asynchronous Status Events:

None

Example:

MOVE SET LOOP 100 TRUE¶

OK{100:MOVESETLOOP}¶

MOVE UNLOCK

The MOVE UNLOCK command unlocks a Move. Moves that are unlocked can be deleted.

Command:

MOVE UNLOCK move_id¶

Arguments:

move_id: Number:

Response:

OK

Context: String: Move ID

Action: String: MOVEUNLOCK

Errors

NOMOVEIDARG

MOVENOTFOUND

Synchronous Status Events:

MOVEUNLOCK

Asynchronous Status Events:

None

Example:

MOVE UNLOCK 10¶

OK{10:MOVEUNLOCK}¶

MOVE VALIDATE

The MOVE VALIDATE command verifies a Move can be executed within the physical limitation of the system. For example, one Keyframe close to another may require too high speeds for the system to attain.

Command:

MOVE VALIDATE move_id¶

Arguments:

move_id: NUMBER:

Response:

```
INFO
   Context: Number: Move ID
   Name/Value Pairs:
         VALID: Boolean:
INFO (1 per error)
   Context: Float: current Keyframe Distance on the move timeline
   Name/Value Pairs:
         AXIS: String:
NEXTKEYFRAME: Float: current position of the next keyframe on the
         timeline, in seconds:
MINIMUMDISTANCE: Float: minimum possible position of this keyframe
         on the timeline, in seconds
OK
   Context: String: Move ID
   Action: String: MOVEVALIDATE
```

Errors

```
NOMOVEIDARG
MOVENOTFOUND
```

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE VALIDATE 3¶

INFO{3:VALID=FALSE, CODE=NOKEYF}

INFO{2.000000:AXIS=ZOOM,NEXTKEYFRAME=3.000000,MINIMUMDISTANCE=3.080000}

INFO{2.000000:AXIS=PAN,NEXTKEYFRAME=3.000000,MINIMUMDISTANCE=4.910000}

INFO{2.000000:AXIS=TILT,NEXTKEYFRAME=3.000000,MINIMUMDISTANCE=3.340000}

INFO{2.000000:AXIS=TRACK,NEXTKEYFRAME=3.000000,MINIMUMDISTANCE=5.040000}

OK{3:MOVEVALIDATE}¶

MOVE ADJUST DISTANCE

The MOVE ADJUST DISTANCE command shifts a keyframe and all subsequent keyframes by a specified amount.

Command:

MOVE ADJUST DISTANCE moveId startKeyframe deltaTime¶

Arguments:

moveId: NUMBER: an integer identifying the move to adjust

startKeyframe: NUMBER: a float identifying the first keyframe to shift

deltaTime: NUMBER: a float representing the duration by which to shift all keyframes at or beyond startKeyframe.

Response:

OK

Context: String: Move ID

Action: String: MOVEADJUSTDISTANCE

Errors

NOMOVEIDARG – Argument moveId not supplied

MOVENOTFOUND – No move exists with moveId

MOVELOCKED – Move is locked and cannot be modified

NOKEYFRAMEARG – Argument startKeyframe not supplied

BADDISTANCEARG – Argument startKeyframe is not a number

NOKEYATDISTANCE – No keyframe exists at timeline distance 'startKeyframe'

NONDELTATIMEARG – Argument deltaTime not supplied

BADDELTATIMEARG - Argument deltaTime is not a number

Synchronous Status Events:

MOVEADJUSTDISTANCE

Asynchronous Status Events:

None

Example:

MOVE ADJUST DISTANCE 3 5.000 1.20¶

OK{3:MOVEADJUSTDISTANCE}¶

MOVE RECORD

The MOVE RECORD command starts the specified Move (if it is CUED), but records any motion applied to the specified axes and replaces the original axes' motions in the Move with the recorded ones.

Command:

MOVE RECORD moveId until recordingTime axesToRecord ¶

Arguments:

moveId: NUMBER: an integer identifying the move to adjust.

until recordingTime (optional): CONSTANT NUMBER: If specified, recording the axes' motions will stop after the given time.

axesToRecord (optional): STRING: A list of axis name strings, use space to separate each axis; if this list is empty all axes will be recorded.

Response:

OK

Context: String: Move ID

Action: String: MOVE RECORD

Errors

NOMOVEIDARG

MOVENOTFOUND

MOVELOCKED

MOVENOTPREPRARED

BEYONDLIMITS

Synchronous Status Events:

MOVEFORWARD

MOVEAXISRECORDINGENABLE

Asynchronous Status Events:

MOVEFORWARDFINISHED

MOVEAXISRECORDINGADDED

Example:

MOVE RECORD 3 UNTIL 11.00 PAN TILT ¶

OK{3:MOVERECORD}¶

MOVE VALIDATE

The MOVE VALIDATE command validates the move.

Command:

MOVE VALIDATE moveId¶

Arguments:

moveId: NUMBER: an integer identifying the move to adjust.

Response:

OK

Context: String: Move ID

Action: String: MOVE VALIDATE

Errors

NOMOVEIDARG

MOVENOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE VALIDATE 3¶

INFO{3:VALID=TRUE}

OK{3:MOVEVALIDATE}¶

MOVE STOP

The MOVE STOP command to stop the specified running move; Furio will return how long it is going to take for the Move to fully stopped.

Command:

MOVE STOP moveId¶

Arguments:

moveId: NUMBER: an integer identifying the move to adjust.

Response:

OK

Context: String: Move ID

Action: String: MOVEVALIDATE

Errors

NOMOVEIDARG

MOVENOTFOUND

MOVENOTEXECUTING

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE STOP 3¶

INFO{3:DISTANCE=0.1}

OK{3:MOVESTOP}¶

MOVE MINIMUM DURATION

The MOVE MINIMUM DURATION command gets the Move's minimum execution duration.

Command:

MOVE MINIMUM DURATION moveId¶

Arguments:

moveId: NUMBER: an integer identifying the move to adjust.

Response:

OK

Context: String: Move ID

Action: String: MOVEMINIMUMDURATION

Errors

NOMOVEIDARG

MOVENOTFOUND

MOVEINVALID

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE MINIMUMDURATION 3¶

*INFO{3:*MOVEMINIMUMDURATION*=2.59}*

OK{3:MOVEMINIMUMDURATION}¶

MOVE FILE SAVE

The MOVE FILE SAVE command to save all Moves.

Command:

MOVE FILE SAVE¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: MOVEFILESAVE

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE FILESAVE¶

OK{:MOVEFILESAVE}¶*

MOVE AXIS RECORDING ENABLE

The MOVE AXIS RECORDING ENABLE command enables specified axis's recording.

Command:

MOVE AXIS RECORDING ENABLE moveId axis¶

Arguments:

moveId: NUMBER: an integer identifying the move

Axis: STRING: axis name

Response:

OK

Context: String: *

Action: String: MOVEAXISRECORDINGENABLE

Errors

NOMOVEIDARG

MOVENOTFOUND

NOAXISARG

MOVELOCKED

RECORDINGNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE AXIS RECORDING ENABLE¶

OK{:MOVEAXISRECORDINGENABLE}¶*

MOVE AXIS RECORDING DISABLE

The MOVE AXIS RECORDING DISABLE command disables specified axis's recording. Which means this axis's recording will not be executed when run this m

Command:

MOVE AXIS RECORDING DISABLE moveId axis¶

Arguments:

moveId: NUMBER: an integer identifying the move

Axis: STRING: axis name

Response:

OK

Context: String: *

Action: String: MOVEAXISRECORDINGDISABLE

Errors

NOMOVEIDARG

MOVENOTFOUND

NOAXISARG

MOVELOCKED

RECORDINGNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

MOVE AXIS RECORDING DISABLE¶

OK{:MOVEAXISRECORDINGDISABLE}¶*

System Commands

CAMERA ID

The CAMERA ID command can be used to determine the unique ID of the Furio head. This ID can for example be used for Thumbnail identification.

Command:

CAMERAID¶

Arguments:

None

Response:

OK

Context: String: Camera ID

Action: String: CAMERAID

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

CAMERAID¶

OK{100-102-401:CAMERAID}¶

NOP

The NOP command replies OK and has no further action. The NOP command can be used by clients to verify the connection with the Furio head is (still) working.

Command:

NOP¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: NOP

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

NOP¶

OK{:NOP}¶*

PREPARED STATUS

The PREPARED STATUS command is used to query if a preset or move is prepared.

Command:

PREPAREDSTATUS¶

Arguments:

None

Response:

```
INFO
   Context: String: *
   Name/Value Pairs:
         NOTHINGPREPARED: String
         MOVEPREPARED: String: move Id
         PRESETPREPARED: String: preset Id
OK
   Context: String: *
   Action: String: PREPAREDSTATUS
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

PREPAREDSTATUS¶

INFO{:NOTHINGPREPARED}*

OK{:PREPAREDSTATUS}¶*

LAST RECALLED STATUS

The LAST RECALLED STATUS command is used to query which preset or move was last recalled, and whether direct control was used to move the robot since the recall.

Command:

LASTRECALLEDSTATUS¶

Arguments:

None

Response:

```
INFO
   Context: String: *
   Name/Value Pairs:
         NOTHINGRECALLED
         LASTPRESETRECALLED: String: preset id
         LASTMOVERECALLED: String: move id
         MOVEDSINCERECALL: Boolean: true if robot moved since last
recall
OK
   Context: String: *
   Action: String: LASTRECALLEDSTATUS
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

LASTRECALLEDTATUS¶

INFO{:LASTPRESETRECALLED=5,MOVEDSINCERECALL=TRUE}*

OK{:LASTRECALLEDSTATUS}¶*

QUIT

The QUIT command cleanly terminates the Control Port sessions. Clients should call this function prior to disconnecting from the Control Port.

Command:

QUIT¶

Arguments:

None

Response:

None

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

QUIT¶

REBOOT

The REBOOT command triggers Furio system to reboot. The communication port will be disconnected right away; there is no response for this command.

Command:

REBOOT¶

Arguments:

None

Response:

None

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

REBOOT¶

SESSION

The SESSION command returns a unique Session ID used to identify the current Control Port session. The Session ID is used on the Status Port to identify the source of an event. Clients can use this command to find out their own Session ID and ignore events generated by their own actions (if wanted).

Command:

SESSION¶

Arguments:

None

Response:

```
OK
   Context: String: Session ID
   Action: String: SESSION
```

Errors

```
None
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
None
```

Example:

SESSION¶

OK{642640:SESSION}¶

SESSION NAME

The SESSION NAME command sets a user-friendly name for this session. The default session name is “NONAME”

Command:

SESSION NAME <name>¶

Arguments:

```
Name: String: the name for this session (UTF-8, max 120 bytes).
```

Response:

```
OK
   Context: String: Session ID
   Action: String: SESSIONNAME
```

Errors

```
NONAMEARG: if the <name> argument is not provided
```

Synchronous Status Events:

```
None
```

Asynchronous Status Events:

```
None
```

Example:

SESSION NAME Station1¶

OK{642640:SESSIONNAME=Station1}¶

SESSION INFO

The SESSION INFO command returns the caller's session name and id.

The SESSION INFO ACTIVE command returns the controlling session's name and id, which may be different than the caller's session. If there is no active controlling session the session Id will be 0.

Command:

SESSION INFO [ACTIVE]¶

Arguments:

ACTIVE (optional): String: a command modifier requesting the session id & name of the session that has joystick control

Response:

INFO

Context: String: session id

NAME: String: session name

OK

Context: String: Session ID

Action: String: SESSIONINFO

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

Request for the current connection's session Id and name.

SESSION INFO

INFO{642640:NAME=Station 1}¶

OK{642640:SESSIONINFO}¶

Request for the active controlling session's Id and name.

SESSION INFO ACTIVE¶

INFO{642640:NAME=Station 2}¶

OK{642640:SESSIONINFO}¶

Request for the active controlling session's Id and name when there is no active controlling session.

SESSION INFO ACTIVE¶

INFO{0:NAME=NONAME}¶

OK{642640:SESSIONINFO}¶

STOP

The STOP command immediately stops all movement at maximum deceleration.

Implementation note: Prior to version 4.9.100.5634, STOP would also release joystick control. As of version 4.9.100.5634, STOP no longer releases joystick control; an explicit call to SLAVING DISABLE is required. See VERSION command.

Command:

STOP¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: STOP

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

STOP¶

OK{:STOP}¶*

HALT

The HALT command immediately stops all movement at maximum deceleration. Same result as STOP.

Command:

HALT¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: HALT

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

HALT¶

OK{:HALT}¶*

HELP

The HELP command provides details of the correct usage of each command.

Command:

HELP¶

Arguments:

None

Response:

Syntax of the asking command, string

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

HELP HALT¶

Syntax: HALT¶

SYSTEM INFO

The SYSTEM INFO command returns a set of properties pertinent to this Furio head.

Command:

SYSTEM INFO¶

Arguments:

None

Response:

INFO

```
   Context:  String: *
   Name/Value Pairs:
         PROTOCOLVERSION: String:
         DEVICE: String: The device type
         STATUSPORT: String: The TCP port of the status channel
         SLAVINGADDRESS: IP Address: Destination for UDP joystick
control.
         SLAVINGPORT: UDP Port: Destination port for UDP joystick
control.
         SHOTCAPACITY: Number: What is the upper limit of shots?
         ONAIRCUT: Boolean: Does robot allow on-air CUTs when
tallied?
         DYNAMICRUNTIME: Boolean: Does robot support time dilation?
         STORELIMITS: Boolean: Does robot support the store limits
workflow?
         NOPRESETFOLLOW: Boolean: Future
         NOPRESETACCELDECEL: Boolean: Does robot support accel
tweaking?
         NOMOVES: Boolean: Does the robot support Moves?
         NOKEYFRAMES: Boolean: Does the robot support Keyframes?
         NOENABLEDISABLEAXIS: Boolean: Can axis be manually
dis/enabled?
         ALTCUEVELOCITY: Boolean: Does robot support AltCueVelocity
feature?
         NODIAGNOSTICS: Boolean: Does robot support a diagnostics
URL?
         NOWEBINTERFACE: Boolean: Does robot have a web interface?
         LASTRECALLED: Boolean: Does robot support LASTRECALLED
      command?
         KEYFRAMETENSION: Boolean: Does robot support tension
            parameter in Move Keyframes (XY axis only)?
         LIMITSVERSION: The version of limits supported
            0 = (HIGH|LO)LIM describe the active Temporary limits
            1 = (HIGH|LO)LIM describe the active limits,
                 with (HIGH|LOW)LIMSOURCE describing the source of
      this limit.
                 PERSISTENT(HIGH|LOW)LIM describes the Persistent
      limits.

         SLAVINGREQUEST: Boolean: True if robot supports "SLAVING
      ENABLE REQUEST" workflow.  See SLAVING ENABLE command.
      Introduced in 6.1.100.
OK
   Context: String: *
   Action: String: SYSTEMINFO
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

SYSTEM INFO¶

INFO{:PROTOCOLVERSION=1.7,DEVICE=CAMBOT,STATUSPORT=13005,SLAVINGADDRESS=172.17.1.53,SLAVINGPORT=10241,SHOTCAPACITY=0,ONAIRCUT=NO,DYNAMICRUNTIME=YES,STORELIMITS=YES,NOPRESETFOLLOW=NO,NOPRESETACCELDECEL=YES,NOMOVES=YES,NOKEYFRAMES=YES,NOENABLEDISABLEAXIS=NO,ALTCUEVELOCITY=YES,NODIAGNOSTICS=YES,NOWEBINTERFACE=YES,KEYFRAMETENSION=YES,SLAVINGREQUEST=YES}¶*

OK{:SYSTEMINFO}¶*

VERSION

The VERSION command returns the firmware version of the Furio head.

Command:

VERSION¶

Arguments:

None

Response:

```
INFO
   CONTEXT: String: *
   Name/Value Pairs:
         VERSION: String: Furio head Firmware Version
OK
   Context: String: *
   Action: String: VERSION
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

VERSION

INFO{:VERSION=4.8.300.5094}*

OK{:VERSION}*

XY Pedestal Specific Commands

The Robotics Server product supports an extended command set that allows control of CamBot XY pedestals. CamBot XY pedestals that natively run Furio firmware and Artimo pedestals also support these commands. Mostly these commands deal with the requirements of a pedestal-based system as opposed to a system that runs on a track.

PED FACE

The PED FACE command turns the pedestal so it faces in a particular direction.

Command:

PED FACE direction [ALTVELOCITY]¶

Arguments:

direction: Number: the direction to face in degrees. Depending on the PED PANRELATIVE command this parameter can be absolute (zero degrees is towards the set, i.e. aligned with the target) or relative to the direction the camera is facing.

ALTVELOCITY (optional): Constant: Use pre-configured alternate velocity limit when executing the motion.

Response:

OK

Context: String: *

Action: String: PEDFACE

Errors

ERR{*:PEDFACE=BADPOSITION}

ERR{*:PEDFACE=NOARG}

ERR{*:PEDFACE=MOTORNOTHOMED}

ERR{*:PEDFACE=BUSY}

ERR{*:PEDFACE=BEYONDLIMITS}

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

PED FACE 90¶

OK{:PEDFACE}¶*

PED TURNAROUND

The PED TURNAROUND command rotates the pedestal by 180 degrees towards the centre of the ped rotational range. It is intended for use when the ped reaches its limit of rotation to make it easy for operator to recover. Ped will rotate at full speed unless the ALTVELOCITY option is set.

If current base rotation<=0, ped will rotate in a clockwise direction by 180 degrees.

If current base rotation > 0, ped will rotate in an anti-clockwise direction by 180 degrees

This command was introduced in SmartShell release 6.0f and applies only to peds running Furio Firmware. It is not implemented in the CamBot translation server implementation of the Furio API for peds running CamBot firmware.

Command:

PED TURNAROUND [ALTVELOCITY]¶

Arguments:

ALTVELOCITY (optional): Constant: Use pre-configured alternate velocity limit when executing the motion.

Response:

OK

Context: String: *

Action: String: PEDTURNAROUND

Errors

ERR{*:PEDTURNAROUND=BADPOSITION} – indicates that rotation is not possible without hitting pan limit

ERR{*:PEDTURNAROUND=MOTORNOTHOMED}

ERR{*:PEDTURNAROUND=BUSY} – indicates that an axis is currently in motion

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

PED TURNAROUND¶

OK{:PEDTURNAROUND}¶*

PED TURNAROUND ALTVELOCITY¶

OK{:PEDTURNAROUND}¶*

PED INFO

The PED INFO command returns details of the pedestal state.

Command:

PED INFO¶

Arguments:

None

Response:

```
INFO
   Context: String: *
   Name/Value Pairs:
         PANRELATIVE: Boolean
         ZOOMVARDEFEAT: Boolean
         TRACKMODE: Enumeration: ROTATE or TRACK or VECTOR or JOG
(JOG added in Furio 7.2d)
OK
   Context: String: *
   Action: String: PEDINFO
```

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Examples:

PED INFO

INFO{:PANRELATIVE=NO,ZOOMVARDEFEAT=NO,TRACKMODE=VECTOR}*

OK{:PEDINFO}¶*

PED PANRELATIVE

The PED PANRELATIVE command enables or disables the pan relative mode of the pedestal.

Command:

PED PANRELATIVE enable¶

Arguments:

enable: Boolean

Response:

OK

Context: String: *

Action: String: PEDPANRELATIVE

Error

ERR{*:PEDPANRELATIVE=BADENABLEARG}

Synchronous Status Events:

None

Asynchronous Status Events:

None

Examples:

PED PANRELATIVE

OK{:PEDPANRELATIVE}¶*

PED STEERINGMODE

The PED STEERINGMODE command determines how the pedestal axis (the XY axis) responds to joystick commands (SLAVING SET INPUTS).

Command:

PED STEERINMODE mode¶

Arguments:

mode: Enumeration: ROTATE or TRACK or VECTOR or JOG (Jog added in Furio 7.2d)

Response:

OK

Context: String: *

Action: String: PEDSTEERINGMODE

Errors

INVALIDMODE

ERR{*:PEDSTEERINGMODE=NOARG}

Synchronous Status Events:

None

Asynchronous Status Events:

None

Examples:

PED STEERINGMODE TRACK

OK{:PEDSTEERINGMODE}¶*

PED ZOOMVARDEFEAT

The PED ZOOMVARDEFEAT command enables or disables the “zoomvar defeat” feature of a CamBot head. When true, full displacement of the PAN stick causes full speed displacement of the Pan axis. When false, full displacement of the PAN stick causes displacement of the Pan axis at a speed proportional to the zoom axis position. The further zoomed in you are, the slower the pan movement.

Command:

PED ZOOMVARDEFEAT enable¶

Arguments:

enable: Boolean

Response:

OK

Context: String: *

Action: String: PEDZOOMVARDEFEAT

Errors

INVALIDMODE

Synchronous Status Events:

None

Asynchronous Status Events:

None

Examples:

PED ZOOMVARDEFEAT TRUE

OK{:PEDZOOMVARDEFEAT}¶*

PRESET SET AUTOROTATE

The PRESET SET AUTOROTATE command determines whether the pedestal auto rotates at the end of a preset.

Command:

PRESET SET AUTOROTATE preset_id enable¶

Arguments:

preset_id: Number: The Preset ID of the Preset to alter.

enable: Boolean

Response:

OK

Context: String: Preset ID

Action: String: PRESETSETAUTOROTATE

Errors

NOPRESETIDARG

PRESETNOTFOUND

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

PRESET SET AUTOROTATE 100 TRUE¶

OK{100:PRESETSETAUTOROTATE}

SYSTEM CLEAR LIMITS

The SYTEM CLEAR LIMITS command instructs the CamBot to clear the limits from non-volatile memory and set them back to the widest range possible for each axis.

Command:

SYSTEM CLEAR LIMITS¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: SYSTEMCLEARLIMITS

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

SYSTEM STORE LIMITS¶

OK{*:SYSTEMSTORELIMITS}

SYSTEM STORE LIMITS

The SYTEM STORE LIMITS command instructs the CamBot to store the limits in non-volatile memory so they are preserved after a power down.

Command:

SYSTEM STORE LIMITS¶

Arguments:

None

Response:

OK

Context: String: *

Action: String: SYSTEMSTORELIMITS

Errors

None

Synchronous Status Events:

None

Asynchronous Status Events:

None

Example:

SYSTEM STORE LIMITS¶

OK{*:SYSTEMSTORELIMIT)

Status Port

The Furio head sends status information on the Status Port. These update messages will be sent to each connected client, independently of which client originated the action resulting in the update. Clients on this port do not respond to any messages received.

Event Structure

As a rule, messages that are sent over this connection consist of three parts:

- An EVT record that marks the start of a new event message and uniquely identifies the type of event.
- Zero or more INFO records that contain additional event data.
- An OK record that marks the end of the event message.

Events are always sent atomically. When an EVT record is received all following records up to and including the OK record belong to the same event.

Event Triggering

In many cases an event is triggered because of a command issued by a client on the Control Port. In such a case the event message format will mimic the reply format of the command that caused the event to trigger. This means that parsing event messages is almost identical as parsing the equivalent command reply, the only difference being the EVT record.

For example, assume that some client uses the Control Port to rename a Preset, as follows (we also show here the session command that the client issued when setting up the Control Port connection):

```
SESSION
OK{9999999901:SESSION}
...
PRESET SET CAPTION 12346 Test¶
OK{12346:PRESETSETCAPTION}¶
```

All clients will be notified of this change via the Status Port. The Furio head will send out the following event:

```
EVT{9999999901:PRESETSETCAPTION}¶
OK{12346:PRESETSETCAPTION}¶
```

As one can see, this mirrors closely the reply to the original command with the addition of the EVT record.

The session ID of the Control Port connection that issued the command that triggered the event is reported in the context of the EVT record. A client should always check the session ID and compare it to its own session ID. If the two match, then the event was generated by the client itself and it can generally be ignored.

The ID of the Preset itself can be determined from the context of the OK record that follows. If a client is interested in the Preset with the reported ID it can then use the control port to request the relevant details from the Furio head.

In general, all Control Port commands that affect a change to the internal database of Presets, Moves and Categories will behave in this way. More specifically, these are the following commands:

- CATEGORY ADD
- CATEGORY DELETE
- CATEGORY SET CAPTION
- MOVE ADD
- MOVE CLONE
- MOVE DELETE
- MOVE LOCK

- MOVE DUPLICATE
- MOVE KEYFRAME ADD
- MOVE KEYFRAME DELETE
- MOVE KEYFRAME SET ACCELERATION
- MOVE KEYFRAME SET DECELERATION
- MOVE SET CAPTION
- MOVE SET CATEGORY
- MOVE SET EXTERNAL ID
- MOVE SET LOOP
- MOVE UNLOCK
- PRESET ADD
- PRESET COPY
- PRESET CLONE
- PRESET DELETE
- PRESET LOCK
- PRESET SET ACCELERATION
- PRESET SET CAPTION
- PRESET SET CATEGORY
- PRESET SET DECELERATION
- PRESET SET DURATION
- PRESET SET POSITION
- PRESET SET EXTERNAL ID
- PRESET UNLOCK

Note: Clients should ignore EVT records with an unrecognized ID. This will ensure compatibility with future versions of the Furio head firmware.

In addition, events may also be triggered when the hardware itself changes state. We will discuss these events individually. Some of these types of events are not associated with a session ID in which case the context field of the EVT record will contain a '*'.

Events

This section describes various Furio head events.

Preset Recall Requested

If a Preset recall is requested, then this event is sent. Depending on if the CUT, CUE, or GOTO mode was used the event action differs. This event was added in SmartShell 4.8b.

Example:

```
EVT{9999999901:PRESETCUTREQUESTED}¶
OK{12345:PRESETCUTREQUESTED}¶

EVT{9999999901:PRESETCUEREQUESTED}¶
OK{12345:PRESETCUEREQUESTED}¶

EVT{9999999901:PRESETGOTOREQUESTED}¶
OK{12346:PRESETGOTOREQUESTED}¶
```

Preset Recall Started

If a Preset is recalled, then this event is sent. Depending on if the CUT, CUE, or GOTO mode was used the event action differs. This event may not fire if the robot is already at or near the target position.

Example:

```
EVT{9999999901:PRESETCUT}¶
OK{12345:PRESETCUT}¶

EVT{9999999901:PRESETCUE}¶
OK{12345:PRESETCUE}¶

EVT{9999999901:PRESETGOTO}¶
OK{12346:PRESETGOTO}¶
```

Preset Recall Finished

If the target position of a Preset recall is reached, then this message is sent. Again, the action field reflects the CUE, CUT or GOTO mode.

Example:

```
EVT{*:PRESETCUEFINISHED}¶
OK{12345: PRESETCUEFINISHED}¶
```

```
EVT{*:PRESETCUTFINISHED}¶
OK{12345:PRESETCUTFINISHED}¶

EVT{*:PRESETGOTOFINISHED}¶
OK{12346:PRESETGOTOFINISHED}¶
```

Moved Away From Last Recalled Preset

This message is sent when a robot moves away from the last recalled Preset as a result of Direct Control. The robot must have reached the Preset position successfully for the event to be fired.

Example:

```
EVT{*:PRESETMOVEDAWAY}¶
OK{124:PRESETMOVEDAWAY}¶
```

Move Recall Requested

If a Key Frame recall or Move Execute recall is requested, then this event is sent. Key Frames are always recalled in CUT mode. This event was added in SmartShell 4.8b.

Example:

```
EVT{9999999901:MOVECUTREQUESTED}¶
OK{12345:MOVECUTREQUESTED}¶

EVT{9999999901:MOVEFORWARDREQUESTED}¶
OK{12345:MOVEFORWARDREQUESTED}¶
```

Move Prepare Started

If a Key Frame is recalled, then this event is sent. Key Frames are always recalled in CUT mode.

Example:

```
EVT{9999999901:MOVECUT}¶
OK{12345:MOVECUT}¶
```

Move Prepare Finished

If the target position of a Key Frame recall is reached, then this message is sent.

Example:

```
EVT{*:MOVECUTFINISHED}¶
OK{12345:MOVECUTFINISHED}¶
```

```
EVT{*:MOVEPREPARED}¶
OK{12345:MOVEPREPARED}¶
```

Move Execute Started

If a Move is executed, then this event is sent.

Example:

```
EVT{9999999901:MOVEFORWARD}¶
OK{12345:MOVEFORWARD}¶
```

Move Execute Finished

If the target position of the last Key Frame in a Move is reached, then this message is sent.

Example:

```
EVT{*:MOVEFORWARDFINISHED}¶
OK{12345:MOVEFORWARDFINISHED}¶

EVT{*:MOVECUTFINISHED}¶
OK{12345:MOVECUTFINISHED}¶
```

Move Prepare Cancelled

If a prepared Move becomes unprepared, then this message is sent.

Example:

```
EVT{*:MOVECUTCANCELLED}¶
OK{12345:MOVECUTCANCELLED}¶
```

Move Recall Stopped

If a Move is stopped while recalling, then this message is sent.

Example:

```
EVT{*:MOVEFORWARDSTOPPED}¶
OK{12345:MOVEFORWARDSTOPPED}¶
```

Move Duration Changed

If the total Duration of a Move is altered, then this message is sent. The Info record gives the new total duration of the move. This event represents a persistent change in the total

duration of a move, unlike the Preset Progress event which represents a temporary change in the total duration of a Preset recall.

Example:

```
EVT{7875440:MOVESETDURATION}
INFO{181:DURATION=15.00}
OK{181:MOVESETDURATION}
```

Moved Away From Last Recalled Move

This message is sent when a robot moves away from the last recalled Move as a result of Direct Control. The robot must have completed the move successfully for the event to be fired.

Example:

```
EVT{*:MOVEMOVEDAWAY}¶
OK{181:MOVEMOVEDAWAY}¶
```

Joystick Control Initiated (Direct Control Initiated)

When a client issues the SLAVING ENABLE command this event is sent. The context field of the EVT record is used to report the session ID, which initiated the joystick control. With this event a client can detect if it has lost direct control to another client.

Example:

```
EVT{9999999901:SLAVINGENABLE}¶
OK{*:SLAVINGENABLE}¶
```

Axis Modified

Many commands elicit an Axis Modified event, which generally looks like this:

```
EVT{9999999901:AXISMODIFIED}
OK{XY:AXISMODIFIED}
```

Heartbeat

The Furio head ensures that within a configurable timeout period at least one heartbeat message is sent on this port. This allows clients to detect a broken connection. The timeout period will be configurable in the settings file of the Furio head. The suggested default value is 10 seconds. It is recommended that clients also implement this as a configurable parameter. The heartbeat event looks like this:

```
EVT{*:NOP}
OK{*:NOP}
```

OffAir

The Furio head sends this event when the robot associated with this connection goes off air. The offair event looks like this:

```
EVT{*:OFFAIR}
OK{*:OFFAIR}
```

OnAir

The Furio head sends this event when the robot associated with this connection goes on air. The onair event looks like this:

```
EVT{*:ONAIR}
OK{*:ONAIR}
```

OnPreview

The Furio head sends this event when the robot associated with this connection goes on preview. The onPreview event looks like this:

```
EVT{*:ONPRV}
OK{*:ONPRV}
```

E-Stop Active

The Furio head sends this event when its hardware Emergency Stop button is activated. The Furio will close all connections to its command channel and stop listening for new connections while the E-Stop is active. The status channel remains open. The E-Stop Active event looks like this:

```
EVT{*:ESTOPACTIVE}
INFO{*:CODE=0}
OK{*:ESTOPACTIVE}
```

Older versions of the protocol do not return the INFO line, in which case CODE is assumed to be zero.

CODE=0 signifies that the hardware ESTOP is triggered by the remote or local red ESTOP button on the robot.

CODE=1 signifies that a collision avoidance accessory has detected an object and triggered the hardware ESTOP to prevent a collision.

E-Stop Clear

The Furio head sends this event when its hardware Emergency Stop button is released. The E-Stop Clear event looks like this:

```
EVT{*:ESTOPCLEAR}
OK{*:ESTOPCLEAR}
```

E-Stop Bypassed

On a CamBot XY system, the Furio head sends this event when detects the estop switches are bypassed. The event looks like this:

Example:

```
EVT{*:ESTOPBYPASSED}
OK{XY:ESTOPBYPASSED}
```

Local Mode

The Furio head sends this event when it enters Local Control mode. The local mode event looks like this:

Example:

```
EVT{*:LOCALMODE}
INFO{*:CODE=LOCALMODE,DESCRIPTION=The\.device\.has\.been\.switched\.
to\.Local\.mode.}
OK{*:LOCALMODE}
```

Preset Progress Update

The Furio head sends this event whenever a preset changes total duration as it is recalling. This feature is only supported by CamBots at this time.

Example:

```
EVT{*:PRESETPROGRESS}
INFO{*:TOTALDURATION=10.56,REMAININGDURATION=7.88}
OK{1:PRESETPROGRESS}
```

Home

The Furio head sends this event when an axis is homed (Robotics Server only). The Home event looks like this:

Example:

```
EVT{*:HOME}
OK{XY:HOME}
```

Limit Reached

The Furio head sends this event when an axis reaches its limits. The Limit Reached event looks like this:

Example:

```
EVT{9999999901:LIMITREACHED}
INFO{TRACK:TYPE=PERSISTENT,LIMITBOUND=HIGHER}
OK{TRACK:LIMITREACHED}
```

Example (XY axis, only when SYSTEM INFO returns LIMITSVERSION=0 or LIMITSVERSION =1):

```
EVT{9999999901:LIMITREACHED}
INFO{XY:TYPE=PERSISTENT,COMPONENT=X,LIMITBOUND=HIGHER}
```

Limit Threshold Reached

The Furio head sends this event when an axis approaches a limit, possibly affecting motion. The Limit Threshold Reached event looks like this:

Example:

```
EVT{9999999901:LIMITTHRESHOLDREACHED}
INFO{TRACK:TYPE=PERSISTENT,LIMITBOUND=HIGHER}
OK{TRACK:LIMITTHRESHOLDREACHED}
```

Example (XY axis):

```
EVT{9999999901:LIMITTHRESHOLDREACHED}
INFO{XY:TYPE=EXTENDED,NAME=Left\.Column.}
OK{XY:LIMITTHRESHOLDREACHED}
```

Warning

The Furio head may send warnings from time to time. These are conveyed using a Warning event which looks like this:

Example:

```
EVT{9999999901:WARNING}
INFO{*:CODE=WARN400,DESCRIPTION=Stop\.Command\.Executed}
OK{*:WARNING}
```

Details:

Error

The Furio head may send errors from time to time. These are conveyed using an Error event which looks like the example below.

Upon receipt of an error with MUSTRECONNECT=YES (or =1), a client should close its command connection and reinitialize.

Multiple INFO records may be returned, indicating that multiple errors occurred at the same time. In this case, the context of the INFO record indicates the faulting axis.

The LATCHEDERROR code may include further information about the cause of the latched error. This additional information is contained in the FAULTCODE attribute.

Examples:

```
EVT{9999999901:ERROR}
INFO{*:CODE=BUMPERHIT,MUSTRECONNECT=YES,DESCRIPTION=The\.pedestal\.bumper\.has\.hit\.an\.obstruction...}
OK{*:ERROR}


EVT{9999999902:ERROR}
INFO{TILT:CODE=LATCHEDERROR,FAULTCODE=PHASINGERROR,DESCRIPTION=Phasing\.error,MUSTRECONNECT=1}
OK{*:ERROR}
```

Joystick Control (Direct Control)

We will now focus on how the control port can be used to directly control the robotics hardware. Normally the hardware can be controlled through a joystick that is managed by the Robotics Control User Interface. The Furio head will only allow one control port connection at a time to control the hardware. This means that it is impossible for more than one joystick to be active at the same moment in time. To request control over the hardware, the client must first set up the joystick control system. Issuing the SLAVING ENABLE command will do a “hostile takeover” of the hardware. The control port that previously had joystick control will lose control over the hardware. When a client loses joystick control, it can detect this by listening for the appropriate event on the Status Port, see also the “Status Port Usage” chapter.

```
SLAVING ENABLE FORCE¶
OK{*:SLAVINGENABLE}¶
```

From now on the robot will respond to joystick control commands on this connection only.

Before actual control is possible each axis needs to be enabled and configured. We will illustrate this with an example using the PAN axis. The example assumes that the robot has just been powered up.

```
ENABLE PAN¶
OK{PAN:ENABLE}¶
HOME PAN¶
OK{PAN:HOME}¶
SET INPUT ANALOGVELOCITY PAN LINEAR 0 MM -500 500¶
OK{PAN:SETINPUTANALOGVELOCITY}¶
```

The first command issued makes sure that the servo for the PAN axis is enabled.

The second command homes the PAN axis.

The third command sets up velocity mode joystick control for the PAN axis on this connection. In this mode, the machine will move at a constant speed, until a new input value is given, or it encounters a limit. The format of the command is specified as follows:

```
SET INPUT ANALOGVELOCITY axis LINEAR start_value MM minimum_value
maximum_value
```

The minimum_value and maximum_value arguments define the range of subsequent SLAVING SET INPUTS commands. The start_value is the initial speed after execution of the SET INPUT ANALOGVELOCITY command. In this example the value -500 is set up as corresponding to the minimum velocity in one direction and the value 500 is set up as the maximum velocity in the other direction.

The following command would start the PAN axis moving at a constant velocity corresponding to 50% of its maximum value in the positive direction:

```
SLAVING SET INPUTS PAN 250¶
OK{PAN:SLAVINGSETINPUTS}¶
```

Note: The meaning of negative or positive numbers in this context is dependent on how the axis is set up. The positive direction can be influenced by the “invert” property of each axis. This property can be changed through the Robotics Control User Interface by the operator. The property can also be changed with the invert command:

```
INVERT PAN¶
OK{PAN:INVERT}¶
```

See other Axis Commands in the Command Overview section, in particular SET RATIO, SET DAMPING, and AXIS SET DYNAMIC RATIO.

UDP Joystick Control

The preceding example illustrates sending joystick commands over the command channel. That method is known as TCP Joystick Control. Starting in SmartShell 4.2, a more responsive "UDP Joystick Control" mechanism was introduced. It should be used for serious implementations.

Note: UDP Joystick Control replaces the "SLAVING SET INPUTS" command only; other joystick control configuration commands must still be sent to configure the UDP joystick control session.

A Furio device that supports UDP Joystick Control will report valid IP address and UDP port in the SLAVINGADDRESS and SLAVINGPORT response arguments of the SYSTEM INFO command. A compliant Furio will then listen for UDP packets conforming to the protocol below.

A stable packet rate of 20 packets/second is ideal. The maximum rate is equal to the video reference rate connected to the head.

Upon receiving a datagram, the robot checks if:

- The session in the packet has joystick control.
- The packet is not out of order (lower time stamp than the previous one received).
- The packet is not stale (it did not arrive later than a certain deadline compared to the arrival of previous one received).

If these conditions hold it applies the joystick control input values.

To handle the case where the UDP Joystick Control emitter dies after sending out a non-zero joystick demand, the robot observes a timeout. If no packets are received within a set timeout, the joystick control session is considered dead and all joystick control input is returned to safe values (i.e. zero for velocity inputs). Subsequent packets are ignored until a new session id is set for joystick input via the status channel.

To see if your UDP Joystick Control emitter has joystick control, send an empty SET SLAVING INPUTS command (no arguments) and look for an OK response. An error response of ERR{*:SLAVINGSETINPUTS=NOTMASTER} indicates the UDP Joystick Control emitter does not have joystick control.

The datagram format is given by this table. In each field the LSB is the lowest-numbered byte.

FieldName	Type	Offset		FieldName	Type	Offset
magic	uint32	0		time_velocity	int32	48
version	uint32	4		x_position	int32	52

timestamp	uint32	8		y_position	int32	56
session	uint32	12		z_position	int32	60
x_velocity	int32	16		pan_position	int32	64
y_velocity	int32	20		tilt_position	int32	68
z_velocity	int32	24		zoom_position	int32	72
pan_velocity	int32	28		focus_position	int32	76
tilt_velocity	int32	32		iris_position	int32	80
zoom_velocity	int32	36		time_position	int32	84
focus_velocity	int32	40		buttons1	uint16	88
iris_velocity	int32	44		buttons2	uint16	90

Field details are given on the next page.

Field Notes	
magic	0x53534F52
version	1
timestamp	Under current Ross implementation this field behaves more like a sequence number than an actual timestamp. It restarts at zero whenever a joystick reboots. A UDP Joystick Control emitter should take that into account, therefore to make it possible to distinguish between a legitimate reset to zero and rogue packet, a reset to zero event must be accompanied by a “SLAVINGENABLE” event.
session	Joystick control session ID obtained from SESSION command.
***_velocity	Velocity demand in the range -128 to +127.
***_position	Position demand in the range Int32.MIN to Int32.MAX. This is simply the integration of the corresponding ***_velocity parameter over time (20 times per second)
buttons1	When set to 1, bit 11 (starting from 0) signals that the time_velocity parameter should be used to dilate (shrink or extend) the total duration of a currently-recalling preset or move. The other bits are reserved for future use.
buttons2	Reserved for future use. Bit 0 is left stick button Bit 1 is right stick button
time_velocity	Negative Time_Velocity corresponds to faster motion and finishes sooner. Positive Time_Velocity corresponds to slower motion and will extend total duration to take longer.

Sample C# source is available to illustrate setting up a UDP joystick control session.

Examples

This series of brief examples are designed to help a first-time implementer get running quickly with a Furio head. On the left are the commands and responses in the command channel, while on the right are the asynchronous events arriving over the status channel.

Opening the Command and Status channels

Command Channel	Status Channel
	<Connect to status port> *EVT{*:ESTOPCLEAR}* *OK{*:ESTOPCLEAR}* *EVT{*:OFFAIR}* *OK{5018001289:OFFAIR}*
<Connect to status port> **CAMERAID** OK{5018001289:CAMERAID}¶ **SESSION**¶ *OK{42:SESSION}¶*	

Enable & Home Axes

A device may power on with its axes disabled by default. Additional commands must be sent before control can be achieved.

AXIS LIST¶ *INFO{PAN:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NOTHOMED}* *INFO{ZOOM:ENABLED=DISABLED,INVERTED=FALSE,HOMING=NA}* *OK{*:AXISLIST}¶*	
ENABLE PAN¶ *OK{PAN:ENABLE}¶*	
	EVT{:AXISMODIFIED}* *INFO{PAN:STATE=HOMING}* *OK{PAN:AXISMODIFIED}*
ENABLE ZOOM¶ *OK{ZOOM:ENABLE}¶*	
	EVT{:AXISMODIFIED}* *INFO{ZOOM:STATE=OPERATIONAL}* *OK{ZOOM:AXISMODIFIED}*
AXIS LIST¶ *INFO{PAN:ENABLED=ENABLED,INVERTED=FALSE,HOMING=NOTHOMED}* *INFO{ZOOM:ENABLED=ENABLED,INVERTED=FALSE,HOMING=NA}* *OK{*:AXISLIST}¶*	

HOME PAN¶

OK{PAN:HOME}¶

<wait>

AXIS LIST¶

INFO{PAN:ENABLED=ENABLED,INVERTED=FALSE,HOMING=HOMING}

INFO{ZOOM:ENABLED=ENABLED,INVERTED=FALSE,HOMING=NA}

OK{:AXISLIST}¶*

<wait>

EVT{:AXISMODIFIED}*

INFO{PAN:STATE=OPERATIONAL}

OK{PAN:AXISMODIFIED}

AXIS LIST¶

INFO{PAN:ENABLED=ENABLED,INVERTED=FALSE,HOMING=HOMED}

INFO{ZOOM:ENABLED=ENABLED,INVERTED=FALSE,HOMING=NA}

OK{:AXISLIST}¶*

Configure Direct Control

This example assumes all axes are homed and in good working order.

SLAVING ENABLE FORCE¶

OK{:SLAVINGENABLE}¶*

EVT{42:SLAVINGENABLE}

OK{:SLAVINGENABLE}*

SET INPUT ANALOGVELOCITY PAN 0 MM -200 200¶

OK{PAN:SETINPUTANALOGVELOCITY}¶

SET INPUT ANALOGVELOCITY ZOOM 0 MM -100 100¶

OK{ZOOM:SETINPUTANALOGVELOCITY}¶

SET RATIO PAN 45¶

OK{PAN:SETRATIO}¶

SET RATIO ZOOM 66¶

OK{ZOOM:SETRATIO}¶

SET DAMPING PAN 12¶

OK{PAN:SETDAMPING}¶

SET DAMPING ZOOM 1¶

OK{ZOOM:SETDAMPING}¶

AXIS SET DYNAMIC RATIO PAN 50¶

OK{PAN:AXISSETDYNAMICRATIO}¶

The robot is ready to receive joystick control demand from this control session, either through UDP or TCP joystick control.

Joystick Control Input via UDP Joystick Control

After having configured joystick control per the previous example, joystick control via UDP Joystick Control is possible.

We need to know where to direct the UDP Joystick Control packets. The System Info command has the answer.

```
SYSTEM INFO¶
INFO{*:PROTOCOLVERSION=1.6,DEVICE=VR600,STATUSPORT=10242,SLAVINGADDR
ESS=172.17.1.53,SLAVINGPORT=10241,ONAIRCUT=NO,ALTCUEVELOCITY=YES}¶
OK{*:SYSTEMINFO}¶
```

In this example, UDP Joystick Control packets must be sent to address 172.17.1.53 at port 10241. In the UDP Joystick Control packet,

- the session field must be set to our session ID, 42.
- pan_velocity can be a value between -200 and +200.
- zoom_velocity can be a value between -100 and +100.

To know whether we have joystick control or not, we send an empty TCP Joystick Control command:

```
SLAVING SET INPUTS¶
OK{*:SLAVINGSETINPUTS}¶
```

If we have lost joystick control since our last Slaving Enable command, the response would be:

```
SLAVING SET INPUTS¶
ERR{*:SLAVINGSETINPUTS=NOTMASTER}¶
```

Preset Recall

This example illustrates how to retrieve the presets in a category, CUT to Preset A, and RUN to Preset B in 15 seconds.

Command Channel	Status Channel
CATEGORY LIST¶	
	INFO{3:CAPTION=Morning\.News}¶
	INFO{5:CAPTION=Evening\.News}¶
	OK{*:CATEGORYLIST}¶

PRESET LIST 5¶

INFO{100:CAPTION=Preset\.A,CATEGORY=5,DURATION=10.000000}¶

INFO{101:CAPTION=Preset\.B,CATEGORY=5,DURATION=10.000000}¶

INFO{110:CAPTION=Preset\.C,CATEGORY=5,DURATION=20.000000}¶

OK{:PRESETLIST}¶*

PRESET CUT 100¶

OK{100:PRESETCUT}¶

EVT{42:PRESETCUTREQUESTED}

OK{100:PRESETCUTREQUESTED}

EVT{457:PRESETCUT}

INFO{100:DURATION=0.50}

OK{100:PRESETCUT}

EVT{:PRESETCUTFINISHED}*

OK{100:PRESETCUTFINISHED}

PRESET GOTO 101 15¶

OK{101:PRESETGOTO}¶

EVT{101:PRESETGOTOREQUESTED}

OK{101:PRESETGOTOREQUESTED}

EVT{42:PRESETGOTO}

INFO{101:DURATION=15.00}

OK{101:PRESETGOTO}

<15 seconds later>

EVT{:PRESETGOTOFINISHED}*

OK{101:PRESETGOTOFINISHED}

Change History

Version	Author	Changes	Date
1.7.8	bsharp	Added JOG steering mode. Cleaned up various entries prior to external release.	2025-10-17
1.7.7	sgervais	Added TRACK mode joystick control example.	2025-08-14
1.7.6	sgervais	Updated with changes related to reaching limits introduced in Furio 7.0a	2024-03-01

1.7.5	sgervais	Replaced ‘slaving’ terminology with ‘joystick’ or ‘joystick control’. Doc change only; API commands unchanged.	2023-10-16
1.7.4	dchen	Add SESSION NAME, SESSION INFO, and SLAVING ENABLE REQUEST for 6.1.100	2023-09-28
1.7.3	dchen	Added a lot of missing commands, now it is up to date with Furio 6.0.600	2023-08-17
1.7.2	bsharp	Added PED TURNAROUND command	2023-06-30
1.7.1	sgervais	Add MOVE KEYFRAME ADD <move_id> Document optional ALTVELOCITY argument in commands.	2020-10-13
1.7.0	sgervais	Add BALANCE START and BALANCE STATUS commands. Numerous corrections and clarifications. Add change history section.	2020-02-02
1.6.2	sgervais	Add an implementation note that 5.0.200 Furio firmware does not support SET INPUT ENCODER (position-based joystick control). Remove deprecated SET INPUT command. Minor clarifications expected/max joystick control input rate. Sample C# source is available that illustrates setting up a UDP Joystick Control client.	2019-08-01
1.6.1	sgervais	Document changes for the SmartShell 4.9 release: - SLAVING DISABLE accepts a session id argument - STOP no longer releases joystick control, must call SLAVING DISABLE - Looped Moves - Last Recalled Status Introduction: document units of measure for each axis. Distinguish whole numbers from floating point numbers by introducing “Float” as a new Data Type. Document MUSTRECONNECT argument in ERROR events. Update TCP Joystick Control documentation, mentioning that homing axes may be necessary, and how to handle that case. Add an Examples section at the end.	2019-07-24
1.5.5	sgervais	Introduction: correct error in Furio Status port. Document TCP Joystick Control of XY axis in each steering mode. Small update to events list. Update command descriptions with additional guidance.	2018-01-26
1.5.3	sgervais	Expanded introduction to more clearly explain the process of	2017-09-

		connecting directly to Furios and to Robotic Server for CamBots. Regroup commands into Axis, Category, Preset, Move, System, and CamBot Specific Commands. General update to account for protocol changes up to and including the SmartShell 4.8 releases: - Connecting to CamBots via Robotics Server (4.0) - Commands added for CamBot integration. (4.0) - UDP Joystick Control (4.2) - Commands related to External IDs (4.2) - Update SYSTEM INFO with new parameters. (4.5) - ESTOP event (4.6) - Update some error codes	08
1.5.2	sgervais		2017-09-05
1.4.9	kkicksee	Reformatted to Ross standards	2015-03-26
1.4.7	kbosmans	Original document from FxMotion	2012-04-23