The iCub CAN protocol for the modern motor boards

This document describes the iCub CAN protocol with focus on the modern motor boards.

Approval History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Version | Author | | Date | Approved | | Date |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.5 | 12 Apr 21 | M. Accame | First WIP version. |
| 0.6 | 27 Nov 23 | J. Losi - V. Gaggero | Added temperature configuration and streaming |
| 0.7 | 25 Sep 24 | M. Accame | Reviewed and completed |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 1](#_Toc178337092)

[2 The messages of the motor polling class 2](#_Toc178337093)

[2.1 Class polling motor control 3](#_Toc178337094)

[2.1.1 Messages used for general management 3](#_Toc178337095)

[2.1.2 Messages used for configuring the motion control 3](#_Toc178337096)

[2.1.2.1 SET\_CONTROL\_MODE 6](#_Toc178337097)

[2.1.2.2 GET\_CONTROL\_MODE 7](#_Toc178337098)

[2.1.2.3 SET\_CURRENT\_LIMIT 8](#_Toc178337099)

[2.1.2.4 GET\_ CURRENT\_LIMIT 9](#_Toc178337100)

[2.1.2.5 SET\_CURRENT\_PID 10](#_Toc178337101)

[2.1.2.6 GET\_CURRENT\_PID 11](#_Toc178337102)

[2.1.2.7 SET\_VELOCITY\_PID 12](#_Toc178337103)

[2.1.2.8 GET\_VELOCITY\_PID 13](#_Toc178337104)

[2.1.2.9 SET\_MOTOR\_CONFIG 14](#_Toc178337105)

[2.1.2.10 GET\_MOTOR\_CONFIG 15](#_Toc178337106)

[2.1.2.11 SET\_TEMPERATURE\_LIMIT 16](#_Toc178337107)

[2.1.2.12 GET\_TEMPERATURE\_LIMIT 17](#_Toc178337108)

[2.1.2.13 SET\_CURRENT\_PIDLIMITS 18](#_Toc178337109)

[2.1.2.14 GET\_CURRENT\_PIDLIMITS 19](#_Toc178337110)

[2.1.2.15 SET\_VELOCITY\_PIDLIMITS 20](#_Toc178337111)

[2.1.2.16 GET\_VELOCITY\_PIDLIMITS 21](#_Toc178337112)

[2.1.2.17 CALIBRATE\_ENCODER 22](#_Toc178337113)

[3 The messages of the motor streaming class 23](#_Toc178337114)

[3.1 Class periodic motor control 23](#_Toc178337115)

[3.1.1 Messages for debug 23](#_Toc178337116)

[3.1.2 Messages which send setpoints 24](#_Toc178337117)

[3.1.2.1 EMSTO2FOC\_DESIRED\_CURRENT, but better DESIRED\_TARGETS 25](#_Toc178337118)

[3.1.3 Messages which stream status 26](#_Toc178337119)

[3.1.3.1 2FOC 27](#_Toc178337120)

[3.1.3.2 STATUS 28](#_Toc178337121)

[3.1.3.3 ADDITIONAL\_STATUS 31](#_Toc178337122)

[4 Example of messages 32](#_Toc178337123)

[4.1 Some messages on binary format 32](#_Toc178337124)

[4.1.1 Polling 32](#_Toc178337125)

[4.1.2 Broadcasting 35](#_Toc178337126)

[5 Use of the messages inside the ems and foc boards 38](#_Toc178337127)

[5.1 Configuration of the motor 38](#_Toc178337128)

[5.2 Control mode 39](#_Toc178337129)

[5.3 Runtime behaviour 40](#_Toc178337130)

[5.4 Other messages 41](#_Toc178337131)

[6 References 42](#_Toc178337132)

# Introduction

This document describes the iCub CAN protocol specifically used by the modern MC CAN boards. They are boards such as foc, amcbldc, amc, amcfoc.

The foc board is one half of the 2foc PCB. This PCB has two independent MPUs each with a different CAN address and each that manages a single motor.

The amcbldc is a CAN board that manages one BLDC motor and has a single CAN address.

The amc is a dual core board where one core talks to yarprobotinterface and other core the manages one single BLDC motor. They use CAN protocol to communicate with each other via CAN bus or internally via inter-core communication (ICC). The core used for MC uses a single address.

Also the amcfoc is a dual core board where one core talks to yarprobotinterface and the second core manages MC, this time for up to two motors. The CAN protocol is used only over inter-core communication and the MC core uses two CAN addresses, each one dedicated to a single motor.

As a result of that, in modern MC CAN boards each CAN address manages only one motor.

The modern MC CAN boards are opposed to legacy MC CAN boards such as 2bll and 4dc (also named mc4can). They can manage up to two motors per CAN address. They also use a number of CAN messages of motor polling class that we shall not treat.

For more information about the iCub CAN protocol, addressing, classes etc., refer to the document *TSD-ICUB-CAN-protocol-sensor-boards* (see [1])

The messages of the motor polling class

Here is description of the messages of the motor polling class managed by the modern MC boards.

## Class polling motor control

The messages of class polling motor control (CLS = 000b) are used mostly for motor control boards (MOT). However, some messages of this class have a general use and are also supported by the sensor board.

### Messages used for general management

They are messages originally developed for MC boards that have similar versions for sensor boards.

They are the messages with CMD in the following table.

|  |  |
| --- | --- |
| CMD | Description |
|  | … others |
| 0x32 | SET\_BOARD\_ID |
|  | … others |
| 0x52 | GET\_FIRMWARE\_VERSION |
|  | … others |

**Table 1** – The CMDs of the polling motor control class (supported by some sensor boards).

To avoid duplications of information, for details refer to the document TSD-ICUB-CAN-protocol-sensor-boards which originally treated them.

### Messages used for configuring the motion control

These messages are used to configure the motion control service in boards which manage one or two motors.

The protocol rules in class polling motor control tell that each board can manage at most two motors. The info of the motor number is contained in the most significant bit of the field CMD. As such, this rule limits the possible commands specified by the CMD field to 128 values from 0x00 to 0x7F.

However, modern CAN motor boards manage one single motor per address, so all the messages will always refer to the first motor.

The messages we use for the modern MC boards are a small subset of the complete set of CAN messages of the class polling motor control.

In here is a table of all the messages which are supported by the modern MC boards with some comments.

|  |  |
| --- | --- |
| CMD | Description |
| 7 | GET\_CONTROL\_MODE  It is not managed by foc / amcbldc / amc. |
| 9 | **SET\_CONTROL\_MODE**  Sets the control mode |
| 124 | GET\_CURRENT\_LIMIT  It is not managed by foc / amcbldc / amc. |
| 72 | **SET\_CURRENT\_LIMIT**  Sets the current limits |
| 101 | **SET\_CURRENT\_PID**  It transport the three PID gains, but not the other values that are transported by SET\_CURRENT\_PIDLIMITS which apparently is not used by any modern motor board. |
| 102 | GET\_CURRENT\_PID  It is not managed by foc / amcbldc / amc |
| 103 | SET\_CURRENT\_PIDLIMITS  It is not managed |
| 104 | GET\_CURRENT\_PIDLIMITS  It is not managed |
| 105 | **SET\_VELOCITY\_PID**  It transport the three PID gains, but not the other values that are transported by SET\_VELOCITY\_PIDLIMITS which apparently is not used by any modern motor board. |
| 106 | GET\_VELOCITY\_PID  It is not managed by foc / amcbldc / amc |
| 107 | SET\_VELOCITY\_PIDLIMITS  It is not managed |
| 108 | GET\_VELOCITY\_PIDLIMITS  It is not managed |
| 119 | **SET\_MOTOR\_CONFIG**  This message is only for the modern MC boards. |
| 123 | GET\_MOTOR\_CONFIG  This message is only for the modern MC boards. It is not managed by foc / amcbldc / amc. |
| 121 | **SET\_TEMPERATURE\_LIMIT**  This message is only for the modern MC boards. It is not managed by the amcbldc, amc. |
| 122 | GET\_TEMPERATURE\_LIMIT  This message is only for the modern MC boards. It is not managed by the foc, amcbldc, amc. |
| 4 | CALIBRATE\_ENCODER  I suspect that the command is not managed correctly by the pair ems / 2foc because the function in Motor.c which use passes a NULL pointer instead of the arguments expected by the iCub CAN protocol.  As a result of that, I suspect that an undefined calibration type is used.  This command is not managed by amcbldc, amc, amcfoc. |

**Table 2** – The CMDs of the polling motor control class managed by the modern MC boards.

#### SET\_CONTROL\_MODE

|  |  |
| --- | --- |
| CMD | Description |
| **0x09**  **0x89** | **SET\_CONTROL\_MODE**  **In brief**  Used to impose a control mode to the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**    M|0x09  CTRL   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 1. * CTRL is the required control mode.   Its value is summarised by enum class embot::prot::can::motor::ControlMode that picks up the values inside icubCanProto\_controlmode\_t that are effectively used:  ControlMode::Idle = 0x00, used both as command to be in idle state or for reporting it;  ControlMode::ForceIdle = 0x09, that forces the idle state;  ControlMode::Current = 0x06, for current control mode;  ControlMode:: SpeedVoltage = 0x0A, for velocity control mode;  ControlMode::OpenLoop = 0x50, for voltage control mode;  ControlMode:: NotConfigured = 0xB0, to report not configured state;  ControlMode:: HWfault = 0xA0, to report an HW fault state.  **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It applies the control mode to motor m.  **Reply**    app<MC>  It does not reply. |
|  |  |

#### GET\_CONTROL\_MODE

|  |  |
| --- | --- |
| CMD | Description |
| **0x07**  **0x87** | **GET\_CONTROL\_MODE**  **In brief**  It asks the control mode for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**    M|0x07   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It sends back the control mode of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x07  CTRL   * M is the same as received. * sizeof(ARG) = 1. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_CURRENT\_LIMIT

|  |  |
| --- | --- |
| CMD | Description |
| **0x48**  **0xC8** | **SET\_CURRENT\_LIMIT**  **In brief**  Used to configure current limits of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**    M|0x48  X  NOMINAL  PEAK  OVERLOAD   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * X is an undocumented and un-used value (the ems board uses 0). * NOMINAL is the nominal current expressed in mA units contained inside a int16\_t with little endian ordering. * PEAK is the peak current expressed in mA units contained inside a int16\_t with little endian ordering. * OVERLOAD is the overload current expressed in mA units contained inside a int16\_t with little endian ordering.   **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It applies the limits to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The ems board uses this message when treating data inside eOmc\_current\_limits\_params\_t. |

#### GET\_ CURRENT\_LIMIT

|  |  |
| --- | --- |
| CMD | Description |
| **0x7C**  **0xFC** | **GET\_ CURRENT\_LIMIT**  **In brief**  It asks the current limits for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**    M|0x7C   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It sends back current limits of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x7C  X  NOMINAL  PEAK  OVERLOAD   * M is the same as received. * sizeof(ARG) = 7. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_CURRENT\_PID

|  |  |
| --- | --- |
| CMD | Description |
| **0x65**  **0xE5** | **SET\_CURRENT\_PID**  **In brief**  Used to configure the current PID of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**    M|0x65  KP  KI  KD  KS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * KP is the proportional gain expressed in int16\_t with little endian ordering. * KI is the integral gain expressed in int16\_t with little endian ordering. * KD is the derivative gain expressed in int16\_t with little endian ordering. * KS is the shift factor expressed in uint8\_t w/ range [0, 15].   **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It applies the PID values to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The current PID transforms currents [A] into voltage [V], so the measurement unit of the proportional gain must be [V/A].    The KP is [V/A]. The KI is hence [(V/A)/s] and the KP [(V/A)\*s].  The gains k = (KP, KI, KD) are transformed in floating point values with formula:  v = static\_cast<float>(k) \* pow(2, -KS) |

#### GET\_CURRENT\_PID

|  |  |
| --- | --- |
| CMD | Description |
| **0x66**  **0xE6** | **GET\_ CURRENT\_PID**  **In brief**  It asks the current PID for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  amcfoc.mot  **Format of PAYLOAD**    M|0x66   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  amcfoc.mot  It sends back the current PID of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x66  KP  KI  KD  KS   * M is the same as received. * sizeof(ARG) = 7. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_VELOCITY\_PID

|  |  |
| --- | --- |
| CMD | Description |
| **0x69**  **0xE9** | **SET\_VELOCITY\_PID**  **In brief**  Used to impose the velocity PID of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**  M|0x69  KP  KI  KD  KS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * KP is the proportional gain expressed in int16\_t with little endian ordering. * KI is the integral gain expressed in int16\_t with little endian ordering. * KD is the derivative gain expressed in int16\_t with little endian ordering. * KS is the shift factor expressed in uint8\_t w/ range [0, 15].   **Actions on reception**    appl<MC>  mc-generic  foc  amcbldc  amc.mot  amcfoc.mot  It applies the PID values to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The velocity PID transforms velocity [deg/s] into current [A], so the measurement unit of the proportional gain must be [current / (deg/s)].    This message uses current expressed in mA, so KP contains [mA / (deg/s)]. The KI contains hence [(mA / (deg/s))/s] and the KP [(mA / (deg/s))s].  The gains k = (KP, KI, KD) are transformed in floating point values with formula:  v = static\_cast<float>(k) \* pow(2, -KS) \* 0.001 |

#### GET\_VELOCITY\_PID

|  |  |
| --- | --- |
| CMD | Description |
| **0x6A**  **0xEA** | **GET\_ VELOCITY\_PID**  **In brief**  It asks the velocity PID for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  amcfoc.mot  **Format of PAYLOAD**    M|0x6A   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  amcfoc.mot  It sends back the velocity PID of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x6A  KP  KI  KD  KS   * M is the same as received. * sizeof(ARG) = 7. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_MOTOR\_CONFIG

It sends the motor configuration.

|  |  |
| --- | --- |
| CMD | Description |
| **0x77**  **0xF9** | **SET\_MOTOR\_CONFIG**  **In brief**  Used to configure the motor only of modern MOT boards, so the associated CMD = 0xF7 is not used.  **Parsed by**    appl<MC>  foc  amcbldc  amc.mot  amcfoc.mot  **Format of PAYLOAD**  0x77  FLAGS  ROTENCRES  ROTINDEXOFF  POLES  ENTLR   * sizeof(ARG) = 7. * FLAGS contains the following binary flags: HAS\_QE = 0x01 (quadrature encoder for rotor), HAS\_HALL = 0x02 (hall sensor), HAS\_TSENS = 0x04 (temperature sensor), USE\_INDEX = 0x08 (index for rotor), HAS\_SPEED\_QE = 0x10 (speed rotor encoder), ENA\_VERBOSE = 0x20 (verbose). * ROTENCRES contains the resolution of the rotor encoder, expressed in int16\_t with little endian ordering. * ROTINDEXOFF contains the offset of the rotor encoder, expressed in int16\_t with little endian ordering. * POLES contains the number of poles in the motor, expressed in uint8\_t. * ENTLR contains the encoder tolerance, expressed in uint8\_t.   **Actions on reception**    appl<MC>  foc  amcbldc  amc.mot  amcfoc.mot  It applies the motor configuration.  **Reply**    app<MC>  It does not reply.  **NOTES**  none |

#### GET\_MOTOR\_CONFIG

|  |  |
| --- | --- |
| CMD | Description |
| **0x7B**  **0xFB** | **GET\_ MOTOR\_CONFIG**  **In brief**  It asks the config for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  amcfoc.mot  **Format of PAYLOAD**    M|0x7B   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  amcfoc.mot  It sends back the config of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x66  FLAGS  ROTENCRES  ROTINDEXOFF  POLES  ENTLR   * M is the same as received. * sizeof(ARG) = 7. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_TEMPERATURE\_LIMIT

|  |  |
| --- | --- |
| CMD | Description |
| **0x79** | **SET\_TEMPERATURE\_LIMIT**  **In brief**  Used to configure temperature limits of motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**  M|0x79  HW\_LIMIT   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 2. * HW\_LIMIT is the hardware temperature limit and is expressed in raw value contained inside a int16\_t with little endian ordering. The motor will be damaged above this max temperature. The raw value depends from the sensor used. In foc board they are PT100 or PT1000.   **Actions on reception**    appl<MC>  mc-generic  foc  It applies the limits to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  The amcblcd, amc.mot and amcfoc.mot do not support (yet) this message. |

#### GET\_TEMPERATURE\_LIMIT

|  |  |
| --- | --- |
| CMD | Description |
| **0x7A**  **0xFA** | **GET\_TEMPERATURE\_LIMIT**  **In brief**  It asks the temperature limit for the motor m = 0 or 1.  **Parsed by**    appl<MC>  mc-generic  amcfoc.mot  **Format of PAYLOAD**    M|0x7A   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 0.   **Actions on reception**    appl<MC>  mc-generic  amcfoc.mot  It sends back the temperature limit of motor m.  **Reply**    app<MC>  It send a message w/ format:    M|0x7A  HW\_LIMIT   * M is the same as received. * sizeof(ARG) = 2. * ARG is the same as the SET\_ command. |
|  |  |

#### SET\_CURRENT\_PIDLIMITS

Not managed by modern MC boards.

It should keep PID parameters other than the gains.

#### GET\_CURRENT\_PIDLIMITS

Not managed by modern MC boards.

It should keep PID parameters other than the gains.

#### SET\_VELOCITY\_PIDLIMITS

Not managed by modern MC boards.

It should keep PID parameters other than the gains.

#### GET\_VELOCITY\_PIDLIMITS

Not managed by modern MC boards.

It should keep PID parameters other than the gains.

#### CALIBRATE\_ENCODER

**Not used**, nevertheless we have documentation so we keep it.

|  |  |
| --- | --- |
| CMD | Description |
| **0x04**  **0x84** | **CALIBRATE\_ENCODER**  **In brief**  Used to calibrate the encoder for motor m = 0 or 1. This message models the struct icubCanProto\_calibrator\_t which has a calibration type and some parameters which depends on it.  **Parsed by**    appl<MC>  mc-generic  foc  **Format of PAYLOAD**    M|0x04  CTYPE  CPARS   * M is either 0x00 (when motor m is 0) or 0x80 (when motor m is 1). * sizeof(ARG) = 7. * CTYPE is the type of calibration. Its value is in icubCanProto\_calibration\_type\_t and can assume values such as: type0\_hard\_stops = 0, type1\_abs\_sens\_analog = 1, type2\_hard\_stops\_diff = 2, type3\_abs\_sens\_digital = 3, type4\_abs\_and\_incremental = 4, type6\_mais = 6, type7\_hall\_sensor = 7. * CPARS are the calibration parameters which depend on the calibration type. Here are some examples:   type0\_hard\_stops: contains the struct icubCanProto\_calibrator\_params\_type0\_hard\_stops\_t    PWMLIMIT  VELOCITY  ZEROPAD  - PWMLIMIT is the value pwmlimit expressed in int16\_t mapped in little endian order  - VELOCITY is the velocity expressed in int16\_t mapped in little endian order  - ZEROPAD is a field with value 0x0000.  …  …  **Actions on reception**    appl<MC>  mc-generic  foc  It applies the specified calibration to motor m.  **Reply**    app<MC>  It does not reply.  **NOTES**  This message is **NOT required**, hence **not managed** by modern MC CAN boards. Probably not even foc. |

The messages of the motor streaming class

Here is description of the messages of the motor streaming class.

## Class periodic motor control

The class periodic motor control (CLS = 001b) contains messages used by motor control boards to stream their status to the host or also to other motor control boards.

|  |  |  |
| --- | --- | --- |
| TYP | Description | On modern MC boards |
| 0x0 | 2FOC | OK |
| 0x1 | POSITION |  |
| 0x2 | PID\_VAL |  |
| 0x3 | STATUS | OK |
| 0x4 | CURRENT |  |
| 0x5 | OVERFLOW |  |
| 0x6 | PRINT | OK |
| 0x7 | VELOCITY |  |
| 0x8 | PID\_ERROR |  |
| 0x9 | DEBUG |  |
| 0xA | MOTOR\_POSITION |  |
| 0xB | MOTOR\_SPEED |  |
| 0xC | ADDITIONAL\_STATUS | OK |
| 0xF | EMSTO2FOC\_DESIRED\_CURRENT | OK |

**Table 3** – The TYPs of the periodic motor control class.

Full description of these messages is out of the scope of this document. In here, we shall describe only the messages supported by the modern MC board.

They are: 2FOC, EMSTO2FOC\_DESIRED\_CURRENT, PRINT, STATUS and ADDITIONAL\_STATUS.

### Messages for debug

The message PRINT is supported by the MOT boards and by some other sensor boards. Its full description can be found in document *TSD-ICUB-CAN-protocol-sensor-boards* [1].

### Messages which send setpoints

So far, we have one message only that sends the desired setpoints to the boards.

#### EMSTO2FOC\_DESIRED\_CURRENT, but better DESIRED\_TARGETS

This message sends all over the CAN bus the target for up to four boards.

It should be a polling message sent to each board, but historically we have always used a single one sent to all.

|  |  |
| --- | --- |
| TYP | Description |
| **0xF** | **EMSTO2FOC\_DESIRED\_CURRENT**  **In brief**  It transports the targets for up to four motors, one per board. Each board will extract the target that is associated to its address.  **Emitted by**    ems  amc.yri  amcfoc.yri  Transmission typically happens every 1 ms.  **Format of DATA**    TARGET-1  TARGET-2  TARGET-3  TARGET-4   * sizeof(DATA) = 8. * TARGET-i is the target for the board w/ address i. It is expressed an int16\_t in little endian format and its measure unit depends on the control mode of the i-th board.   In particular, it can contain current, voltage, velocity or (even if not used so far) position.  The measurement units are:   * + Current is in [mA]   + Voltage is in PWM percentage expressed in range [-32000 = -100%, +32000 = +100%]   + Velocity is in [ICUBDEG/ms]   + Position is in [ICUBDEG] that if treated as an int16\_t has a range of [-180, +180).   **NOTE** |

### Messages which stream status

The modern motor boards signal their status with the following messages. They do not have information about which motor they contain data, so we must use them for one motor only.

In case we have two motors on the same board and we use the iCub Can protocol we must use two different CAN addresses.

#### 2FOC

This message sends core information about the motor

|  |  |
| --- | --- |
| TYP | Description |
| **0x0** | **2FOC**  **In brief**  It transports data such as: current, velocity and position.  **Emitted by**    foc  amcbldc  amc.mot  amcfoc.mot  Transmission is triggered by reception of message EMSTO2FOC\_DESIRED\_CURRENT.  **Format of DATA**    CURRENT  VELOCITY  POSITION   * sizeof(DATA) = 8. * CURRENT is an int16\_t in little endian format which contains the values of current measured by the board. Unit is mA. * VELOCITY is an int16\_t in little endian format which contains the values of velocity measured by the board. Unit is ICUBDEG/ms. * POSITION is an int32\_t in little endian format which contains the values of position measured by the board. Unit is ICUBDEG.   NOTE |

#### STATUS

The status of the motor control may depend on the given board (or even motor).

This message was designed specifically for the foc board and was used also for the others. However, it may change in future.

So the content of the fields QES and FAULTS may in future depend on the emitting board

|  |  |
| --- | --- |
| TYP | Description |
| **0x3** | **STATUS**  **In brief**  It transports data such as: control mode, PWM feedback and faults .  **Emitted by**    foc  amcbldc  amc.mot  amcfoc.mot  Transmission is triggered by reception of message EMSTO2FOC\_DESIRED\_CURRENT.  **Format of DATA**    CM  QES  PWM\_FBK  FAULTS   * sizeof(DATA) = 8. * CM is an uint8\_t which contains the control mode of the single motor managed by the board. See message SET\_CONTROL\_MODE for possible values. * QES is an uint8\_t which contains bits that tell the status of the quadrature encoder. * PWM\_FBK is an int16\_t in little endian format which contains the feedback values of the PWM, expressed in percentage mapped in [-32000, +32000]. * FAULTS is a uint32\_t in little endian order which contains flags with the motor faults   NOTE  Content of QES and FAULTS may depend on the board. |

The of QES and FAULTS may depend on the board. In here is how the foc board (and others) uses them.

**The QES**

The QES is managed by object embot::prot::can::motor::board::foc:: QEstate2FOC that defines the following flags:

enum class Flag : uint8\_t

{

dirty = 0,

stuck = 1,

index\_broken = 2,

phase\_broken = 3,

not\_calibrated = 4

};

So, when bit in position 0 of the QES is 1 the encoder is dirty and the value of QES is 0x01

**The FAULTS**

The FAULTS is managed by object embot::prot::can::motor::board::MotorFaultState2FOC that defines the following flags:

enum class Flag : uint8\_t

{

// they end up in B0

ExternalFaultAsserted = 7,

UnderVoltageFailure = 6,

OverVoltageFailure = 5,

OverCurrentFailure = 4,

DHESInvalidValue = 3,

AS5045CSumError = 2,

DHESInvalidSequence = 1,

CANInvalidProtocol = 0,

// they end up in B1

CAN\_BufferOverRun = 15,

SetpointExpired = 14,

CAN\_TXIsPasv = 13,

CAN\_RXIsPasv = 12,

CAN\_IsWarnTX = 11,

CAN\_IsWarnRX = 10,

OverHeatingFailure = 9,

unused = 8,

// they end up in B2

ADCCalFailure = 23,

I2TFailure = 22,

EMUROMFault = 21,

EMUROMCRCFault = 20,

EncoderFault = 19,

FirmwareSPITimingError = 18,

AS5045CalcError = 17,

FirmwarePWMFatalError = 16,

// they end up in B3

CAN\_TXWasPasv = 31,

CAN\_RXWasPasv = 30,

CAN\_RTRFlagActive = 29,

CAN\_WasWarn = 28,

CAN\_DLCError = 27,

SiliconRevisionFault = 26,

PositionLimitUpper = 25,

PositionLimitLower = 24

};

So that bit in position 7 of the uint32\_t word FAULTS (that is aligned in little endian) contains information about ExternalFaultAsserted.

To clarify:

Content of the field FAULTS for the foc board

1

Bit 14: SetpointExpired = 1

Bit 0: CANInvalidProtocol = 1

FAULTS = 0x80004081

Bit 7: ExternalFaultAsserted = 1

Bit 8: unused = 0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

1

0

0

0

0

0

0

1

0

0

0

0

0

0

1

FAULTS = 10000000000000000100000010000001b = 0x80004081

0x80

0x40

0x81

Bit 31: CAN\_TXWasPasv = 1

0x00

**Figure 1**: The byte by byte content of FAULTS. The content of the mask when mapped in memory can be tricky because the order of the uint32\_t is little endian.

#### ADDITIONAL\_STATUS

So far it transports only temperature. It may change in the future.

|  |  |
| --- | --- |
| TYP | Description |
| **0xC** | **ADDITIONAL\_STATUS**  **In brief**  It transports data from the foc board: current motor temperature retrieved from temperature sensor.  **Emitted by**  foc  Transmission happens from the foc board every 100+board\_id\*100 ms and then parsed by ems in MController.  **Format of DATA**    TBD  TEMPERATURE   * sizeof(DATA) = 4. * TBD are two bytes so far not managed * TEMPERATURE is an int16\_t in little endian format which contains the value of motor temperature measured by the foc board. Unit is raw.   NOTE  A previous version of this document reported for the TBD field: *icubCanProto\_interactionmode\_t. The first 2 bytes are not managed by the foc board and therefore not overridden. Not only the first byte is skipped but 2 in order to not having parsing problems since temperature is int16\_t.* |

# Example of messages

In here are examples of the binary format of some messages.

Ideally we should place in here one (some) use case for the control of a modern board such as the amcbldc.

## Some messages on binary format

**NOTE**: at date of 27 Sept 2024 their content could be reviewed and better described.

### Polling

FRAME-ctrlmode: command that imposes the IDLE control mode

ID

This frame imposes the IDLE (0x00) control mode to motor m = 0 of the board.

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00 ]

SET\_CONTROL\_MODE = 0x09

sizeof(ARG) = 1

**Figure 2**: The frame that imposes control mode IDLE to motor m = 0 of board at address 1: [0x001 | 0x09, 0x00]. For motor m =1 the message would be: [0x001 | 0x89, 0x00].

FRAME-currlim: command that imposes the current limits

ID

This frame imposes currents to motor 0 in this way:

nominal = 0x0102 (258 mA), peak = 0x0203 (515 mA), overload = 0x0304 (772 mA)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00 ] [0x02, 0x01] [0x03, 0x02] [0x04, 0x03]

SET\_CURRENT\_LIMIT = 0x48

sizeof(ARG) = 7

**Figure 3**: The frame that imposes current limits to motor m = 0 of board at address 1: [0x001 | 0x48, 0x00, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03].

FRAME-currpid: command that imposes the current PID

ID

This frame imposes a PID to motor m = 0 in this way:

Kp = 0x0102 (258), Ki = 0x0203 (515), Kd = 0x0304 (772), Ks = 0x00 (0)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x02, 0x01] [0x03, 0x02] [0x04, 0x03] [0x00]

SET\_CURRENT\_PID = 0x65

sizeof(ARG) = 7

**Figure 4**: The frame that imposes the current PID of motor m = 0 of board at address 1: [0x001 | 0x65, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03, 0x00].

FRAME-velpid: command that imposes the velocity PID

ID

This frame imposes a PID to motor m = 0 in this way:

Kp = 0x0102 (258), Ki = 0x0203 (515), Kd = 0x0304 (772), Ks = 0x01 (1)

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x02, 0x01] [0x03, 0x02] [0x04, 0x03] [0x01]

SET\_VELOCITY\_PID = 0x69

sizeof(ARG) = 7

**Figure 5**: The frame that imposes the velocity PID of motor m = 0 of board at address 1: [0x001 | 0x69, 0x02, 0x01, 0x03, 0x02, 0x04, 0x03, 0x01].

FRAME-setmotorconfig: command that configures the foc

ID

This frame impose: HAS\_QE, HAS\_HALL, rotor encoder resolution = 0x0201 = 513, rotor index offset = 0x0403 = 1027, number of poles = 0x06 = 6, encoder tolerance = 0x07 = 7

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[0x03] [0x01, 0x02] [0x03, 0x04] [0x06] [0x07]

SET\_MOTOR\_CONFIG = 0x77

sizeof(ARG) = 7

**Figure 6**: The frame that imposes the settings of the motor at address 1: [0x001 | 0x77, 0x03, 0x01, 0x02, 0x03, 0x04, 0x06, 0x07].

FRAME-currlim: command that imposes the temperature limit

ID

This frame imposes Temperature limit of 0x15B expressed in raw value to motor 0.

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x15B ]

SET\_TEMPERATURE\_LIMIT = 0x79

sizeof(ARG) = 2

**Figure 7**: The frame that imposes motor temperature limits to motor m = 0 of board at address 1: [0x001 | 0x79, 0x15B].

FRAME-calibencoder: command that start a calibration procedure

ID

This frame imposes a calibration type 0 (hard stop) to motor m = 0 of the board with PWM 0x0100 (256) and velocity 0x0101 (257).

{CLS = POLLING-MC = 000b} {SRC = 0000b} {DST = 0001b} = 0x001

CMD

ARG

[ 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00 ]

CALIBRATE\_ENCODER = 0x04

sizeof(ARG) = 7

**Figure 8**: The frame that starts a calibration of type 0 hard stops to motor m = 0 of board at address 1.

### Broadcasting

FRAME-2FOC: status values

ID

Explanation of the content of DATA.

* DATA[0, 1] contains the current expressed as int16\_t
* DATA[2, 3] contains the velocity expressed as int16\_t
* DATA[4, 5, 6, 7] contains the position expressed as int32\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0001b} {TYP = 2FOC = 0x0} = 0x110

DATA

[LSB-CUR] [MSB-CUR] [LSB-VEL] [MSB-VEL] [B0-POS] [B1-POS] [B2-POS] [B3-POS]

sizeof(DATA) = 8

**Figure 9**: This frame contains the values measured and emitted by the foc board at address 0x1: [0x110 | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--]. The frame emitted by board at address 0x2 would have ID = 0x120.

FRAME-2FOC: status values

ID

Explanation of the content of DATA.

* DATA[0] contains the interaction mode for joint 1 and joint 2 expressed as uint8\_t (one per nibble)
* DATA[2, 3] contains the temperature expressed as int16\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0001b} {TYP = 2FOC = 0x0} = 0x110

DATA

[INTMD-J2|INTMD-J1] [LSB-TEMP] [MSB-TEMP]

sizeof(DATA) = 4

**Figure 10**: This frame contains the values measured and emitted by the foc board at address 0x1: [0x110 | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--]. The frame emitted by board at address 0x2 would have ID = 0x120.

FRAME- EMSTO2FOC\_DESIRED\_CURRENT: desired currents as requested by the ems

ID

Explanation of the content of DATA.

* DATA[0, 1] contains the current requested to board w/ address 0x1 expressed as int16\_t
* DATA[2, 3] contains the current requested to board w/ address 0x2 expressed as int16\_t
* DATA[4, 5] contains the current requested to board w/ address 0x3 expressed as int16\_t
* DATA[6, 7] contains the current requested to board w/ address 0x4 expressed as int16\_t

{CLS = PERIODIC-MC = 001b} {SRC = 0000b} {TYP = EMSTO2FOC\_... = 0xF} = 0x10F

DATA

[LSB-C1] [MSB-C1] [LSB-C2] [MSB-C2] [LSB-C3] [MSB-C3] [LSB-C4] [MSB-C4]

sizeof(DATA) = 8

**Figure 11**: This frame contains the values of current that the ems request to all the foc boards. As the frame is emitted by the ems which has CAN address 0x0, the ID is always 0x10F. A typical frame is: [0x10F | 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--, 0x--].

Use of the messages inside the ems and foc boards

Here is an analysis of how the messages are used inside the ems and the foc board.

## Configuration of the motor

This section deals of messages which flow from the ems towards the foc board and are used to configure some motor parameters before any movement of the motor.

The ETH message handler MController\_config\_motor() inside the ems board is called when yarprobotinterface sends the configuration of the whole motor inside struct eOmc\_motor\_config\_t.

This handler calls function MController\_config\_motor() which in turns calls Motor\_config() which fills data inside the MController and then calls Motor\_config\_2FOC() which sends several CAN frames to the foc board:

* SET\_CURRENT\_PID from inside Motor\_config\_current\_PID\_2FOC();
* SET\_VELOCITY\_PID from inside Motor\_config\_velocity\_PID\_2FOC();
* SET\_CURRENT\_LIMIT from inside Motor\_config\_2FOC();
* SET\_MOTOR\_CONFIG from inside Motor\_config\_2FOC();
* SET\_TEMPERATURE\_LIMIT from inside Motor\_config\_2FOC();

Some of the above CAN frames can be transmitted also separately when triggered by other ETH handlers or inside the MController. However, this behaviour mode may not be used by yarprobotinterface which always use the global configuration mode. PLS VERIFY.

Details in the following table.

|  |
| --- |
| CMD |
| **SET\_CURRENT\_PID**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_current\_PID\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the current PID alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_pidcurrent(). |
| **SET\_VELOCITY\_PID**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_velocity\_PID\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the velocity PID alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_pidspeed (). |
| **SET\_CURRENT\_LIMIT**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the current limits alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_currentlimits(). However, in this case the CAN frame is not formed correctly both inside the handler and inside Motor\_config\_max\_currents\_2FOC(). |
| **SET\_MOTOR\_CONFIG**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the resolution of the encoder alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_rotorencoder(). This function calls Motor\_config\_encoder() which effectively transmits the CAN frame. |
| **SET\_TEMPERATURE\_LIMIT**  The message is sent by the ems to the foc inside Motor.c, function Motor\_config\_2FOC() and is triggered by the ETH message handler eoprot\_fun\_UPDT\_mc\_motor\_config() which configures the entire motor.  It can also be triggered by another ETH message handler in case yarprobotinterface wanted to configure the temperature limit alone: eoprot\_fun\_UPDT\_mc\_motor\_config\_temperaturelimitconfig().  This function calls Motor\_config\_max\_temperature(), which then calls Motor\_config\_motor\_max\_temperature\_2FOC() that does the same operations as in Motor\_config\_2FOC() for the temperature part.  If the temperature exceeds the temperature limits the 2FOC board set the motor in fault. |

**Table 4** – The CMDs used for configuration of the foc.

**NOTE**: there are not messages which reads back the values hopefully written in the foc board.

## Control mode

This section deals of messages which flow from the ems towards the foc board and are used change the control mode.

It is used only one command, the SET\_CONTROL\_MODE.

This command is sent by the MController whenever it is necessary to set the motor idle or in the different running modes.

The message is sent by function Motor\_set\_control\_mode\_2FOC() inside Motor.c and in turn is called by Motor\_set\_idle(), Motor\_force\_idle() and Motor\_set\_run(). All the above are called by JointSet.c and Calibrators.c.

|  |
| --- |
| CMD |
| **SET\_CONTROL\_MODE**  The message is sent by the ems to the foc from function Motor\_set\_control\_mode\_2FOC() inside Motor.c  It uses modes such as: icubCanProto\_controlmode\_openloop, icubCanProto\_controlmode\_speed\_voltage or icubCanProto\_controlmode\_current, icubCanProto\_controlmode\_idle, icubCanProto\_controlmode\_forceIdle.  The function Motor\_set\_control\_mode\_2FOC() is sent from several places inside the MController where it is necessary to start o stop the motors (calibration etc). |

**Table 5** – The CMD used to change the control mode.

## Runtime behaviour

During runtime, every millisecond, the ems:

* decodes the CAN frame 2FOC which contains the status of the foc board (current, velocity and position), uses them inside the MController and
* sends the message EMSTO2FOC\_DESIRED\_CURRENT to every foc board with the desired currents.

Here are details.

|  |
| --- |
| CMD |
| **2FOC**  This message is transmitted by the foc to the ems to report its current, velocity and position.  It is parsed by the CAN message handler eocanprotMCperiodic\_parser\_PER\_MC\_MSG\_\_2FOC() at the beginning of every cycle of the control loop and the inside value are:   * copied inside the motor status which is streamed up to yarprobotinterface; * used by the MController with call to function MController\_update\_motor\_odometry\_fbk\_can(). |
| **EMSTO2FOC\_DESIRED\_CURRENT**  This message is sent by the ems to every foc board to impose the desired current. The calling functions is Motor\_actuate() which is in turn called by eo\_motioncontrol\_Tick() every millisecond. |
| **ICUBCANPROTO\_PER\_MC\_MSG\_\_STATUS** This message is transmitted by the foc to the ems to report its system fault and encoder’s fault |
| **ICUBCANPROTO\_PER\_MC\_MSG\_\_ADDITIONAL\_STATUS**  This message is transmitted by the foc to the ems to report motor’s temperature. |

**Table 6** – The messages exchanged between the ems and the foc board at every control cycle.

## Other messages

The message CALIBRATE\_ENCODER is used in function Motor\_uncalibrate() inside Motor.c but the result is very likely a malformed frame because the former function is wrongly passed a NULL pointer to the required parameters.

Here are details.

|  |
| --- |
| CMD |
| **CALIBRATE\_ENCODER**  The message is sent by the ems to the foc from function Motor\_uncalibrate() inside Motor.c  The former of this message is called with wrong parameters and it uses a NULL pointer which may end up in undefined behaviour. The above does not cause any crash of code in the ems but surely produces a malformed CAN frame.  The function Motor\_uncalibrate() is called in cases such as:   * Calibration timeout, function JointSet\_do\_wait\_calibration(), * Start of calibration of tripod or of hands in R1, function JointSet\_calibrate(). |
|  |

**Table 7** – A malformed CAN message which is probably not necessary.

References

[1] TSD-ICUB-CAN-protocol-sensor-boards, <https://github.com/robotology/icub-firmware/blob/master/emBODY/eBdocs/arch-arm/TSD-ICUBUNIT-canprotocol-sensorboards.docx>