iCub Control Modes Specifications

Version	Date	Author	Comments
1.0	17 Jan 2014	marco.randazzo@iit.it	First emission
1.1	25 July 2014	marco.randazzo@iit.it	Minor revision

Contents

1. Introduction

This document describes the control modes made available by YARPdev motor control interfaces and it specifies how they should be implemented. A generic robot must implement the same interfaces and behavioral rules hereby described in order to be compliant with the iCub platform.

General considerations and cautions:

- Control modes described in this document are specific for joint-level control and should be implemented at firmware level by motor control boards. High-level motor skills (e.g. Cartesian interface) are available through other YARP interfaces (not discussed in this document) and rely, at low-level, on the same joint-level controllers.

- YARP interfaces described in this document provide overloaded methods to perform set/get operations on the control modes of:
  - individual joints (e.g. joint 0 of the iCub arm)
  - a sub set of joints belonging to a specific robot part (e.g. joints 0 1 2 of the iCub arm)
  - all the joints belonging to a specific part (e.g. the whole iCub arm)

- A generic robot may be not able to implement all the control modes or interfaces which are made available by YARP. For example, joint-level torque control may be not available on a robot which is not equipped with torque sensors. In this case, the user should be notified by the system that a specific interface is not supported. Because some devices are shared by all robots (for example network proxies) even unavailable interfaces could be implemented. These type of errors can therefore be checked only at runtime.

- A separate issue is how to handle messages when the "wrong" control mode is enabled and how to behave when the user requires a control mode that is not supported. Because some commands are mono-directional and do not receive data back from the boards, there is no safe way to notify the user if commands (reference position, velocity etc.) are invoked when the invalid control mode is enabled. Wrong commands will therefore be only ignored by the boards.

- An error will be returned if the user tries to switch to an invalid control mode (e.g. a control mode not implemented or not available on that specific hardware). Developers are thus recommended to always check the return value of the setControlMode/setInteractionMode methods (Section 4).

2. Control modes and interaction modes

The protocol defines the following control modes, identified by a yarp::os::Vocab code.

1. Position control with trajectory generation[1]	VOCAB_CM_POSITION
2. Direct position control	VOCAB_CM_POSITION_DIRECT
3. Velocity control	VOCAB_CM_VELOCITY
4. Mixed Position-Velocity control	VOCAB_CM_MIXED
5. Torque Control	VOCAB_CM_TORQUE
6. OpenLoop Control	VOCAB_CM_OPENLOOP
7. Idle	VOCAB_CM_IDLE

Table 1: Control Modes

- The selection of a control mode allows the user to select a specific control algorithm to actuate the joint and implicitly defines the accepted commands (i.e. position commands, velocity command etc.).
- Control modes are always explicitly chosen by the user using the proper YARP interfaces: no automatic switch are performed by the system (with the only exceptions of the special board statuses described in Section 3)
- Only the mixed position-velocity control is allowed to accept two command types (i.e. position and velocity commands). All other control modes accept only the command types specific of their interface.
- A warning message is generated if a non-legal command (i.e. a velocity command sent to a joint controlled in position mode) is received by the joint, but no hardware fault condition is generated. The interface may not return an error to the user (for the reasons described in Section 1).

[1] In the iCub trajectories are implemented following a minimum jerk profile, but this is not strictly enforced by this specifications (other robots can implement conventional trapezoidal profile).

In addition to the control modes listed in Table 1, YARP provides two interaction modes, reported in table 2. The selection of the interaction mode will affect only the joint control algorithm, but contrarily to the selection of control modes, this will not alter the type of accepted commands.

8. Stiff Interaction	VOCAB_INTERACTION_STIFF
9. Compliant Interaction	VOCAB_INTERACTION_COMPLIANT

Table 2: Interaction Modes

Stiff interaction is the typical interaction mode of 'industrial' robots, which are required to execute accurate position/velocity trajectories in controlled environments. Using compliant interaction mode, instead, the user can set specific joint impedance (i.e. stiffness σ and damping μ) during the execution of position or velocity commands.

Switching between different control mode and different interaction modes is subject to the following rules:

- The user should be able to switch from stiff interaction mode to compliant interaction mode (and vice-versa), even if a position trajectory has been already commanded. However, changing the interaction mode will stop the joint motion (and the user will have to issue a new position trajectory command).
- If a generic robot is not able to implement the compliant interaction mode (for example because it is not equipped with force/torque sensors), only the stiff interaction mode will be available. If the control board receives an invalid set command, the board mode will automatically switch to a fault status (Section 3.1) and the set method will return an error.
- Idle mode is a "special" mode in which the control algorithm is off and no power is given to the motor. Thus, the user cannot send any motor command to the joint when it is in idle mode. In order to reactivate the controller, the user has to explicitly set a different active control mode before sending a motor command.

The full set possible control modes/interaction modes combinations are summarized in Table 3.

Control Mode	Accepted motor commands	Interaction mode	Additional parameters
Position control with trajectory generation	q	Stiff mode	---
		Compliant mode	σ, μ
Direct position control	q	Stiff mode	---
		Compliant mode	σ, μ
Velocity control	$\dot{q}$	Stiff mode	---
		Compliant mode	σ, μ
Mixed Position-Velocity control	$q, \dot{q}$	Stiff mode	---
		Compliant mode	σ, μ
Torque Control	τ	---	
OpenLoop Control	φ	---	
Idle	---	---	

Table 3: Interaction Modes/ Control Mode combinations

The selection of stiff/compliant interaction mode is meaningful only when the current control mode is 1,2,3 or 4 (i.e. position or velocity control). In all other control modes, such as idle mode, torque mode etc., choosing the interaction mode has no effect on the control, but is accepted and stored in the internal status of the board. It will thus affect the controller when the controller will be again set to control mode which supports the interaction mode. An example of switching between different control modes/interaction modes is shown in Figure 1. Note that the interaction mode is stored in the board also when the controller is idle.

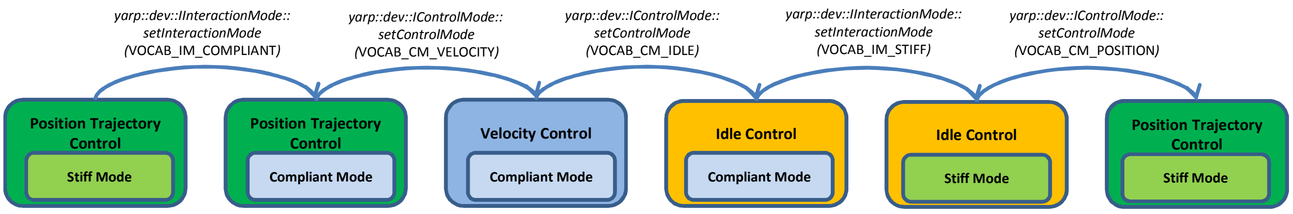
yarp::dev::IInteractionMode:: setInteractionMode (VOCAB_IM_COMPLIANT)
yarp::dev::IControlMode:: setControlMode (VOCAB_CM_VELOCITY)
yarp::dev::IControlMode:: setControlMode (VOCAB_CM_IDLE)
yarp::dev::IInteractionMode:: setInteractionMode (VOCAB_IM_STIFF)
yarp::dev::IControlMode:: setControlMode (VOCAB_CM_POSITION)
Position Trajectory Control
Stiff Mode
Position Trajectory Control
Compliant Mode
Velocity Control
Compliant Mode
Idle Control
Compliant Mode
Idle Control
Stiff Mode
Position Trajectory Control
Stiff Mode

Figure 1: Switching between different control modes/interaction mode

The control algorithm implemented by each control/interaction mode is presented in Figure 2.

- The first row of the picture shows the four position/velocity control modes used with stiff interaction mode. In this case a position controller is used to compute the motor command.
- The second row of the picture shows the same four position/velocity control modes used with compliant interaction mode. In this case an impedance controller computes a torque reference command which is tracked by an inner torque control loop. The impedance controller implements a virtual spring whose stiffness σ and damping factor μ can be continuously set by the user (i.e. without changing control mode).
- The implementation of the velocity control shown in the scheme (i.e. integration of the reference command) corresponds to the current implementation on the iCub 4DC/BLL control boards. However, this is not the only possible implementation: an explicit velocity control loop is also possible.
- The output of the controller φ can be one of the following:
  - Directly the PWM (e.g. iCub BLL/4DC board) if no internal current loop is available.
  - A current reference signal (e.g. 2FOC board). In this case an additional loop is implemented to transform the current reference into the PWM output (current feedback s required).
- Torque control must implement an additional low level check to prevent the movement of the joint against the hardware limit. The same consideration applies also to position/velocity control modes when interaction compliant mode is active. This protection mechanism does not change the currently selected control mode
- In all control modes, an additional feed-forward input, φ_{off} is available.
- The impedance controller used in the compliant interaction mode also allows an additional feed-forward input τ_{off} (which can be used, for example, to compensate gravity).
- Additional mechanisms (not discussed in this document) can be implemented in the position/torque controllers to compensate motors back-EMF and friction.

- Default values for control/interaction modes are defined by the calibration procedure of the joint (see section 3.2 and section 3.3).

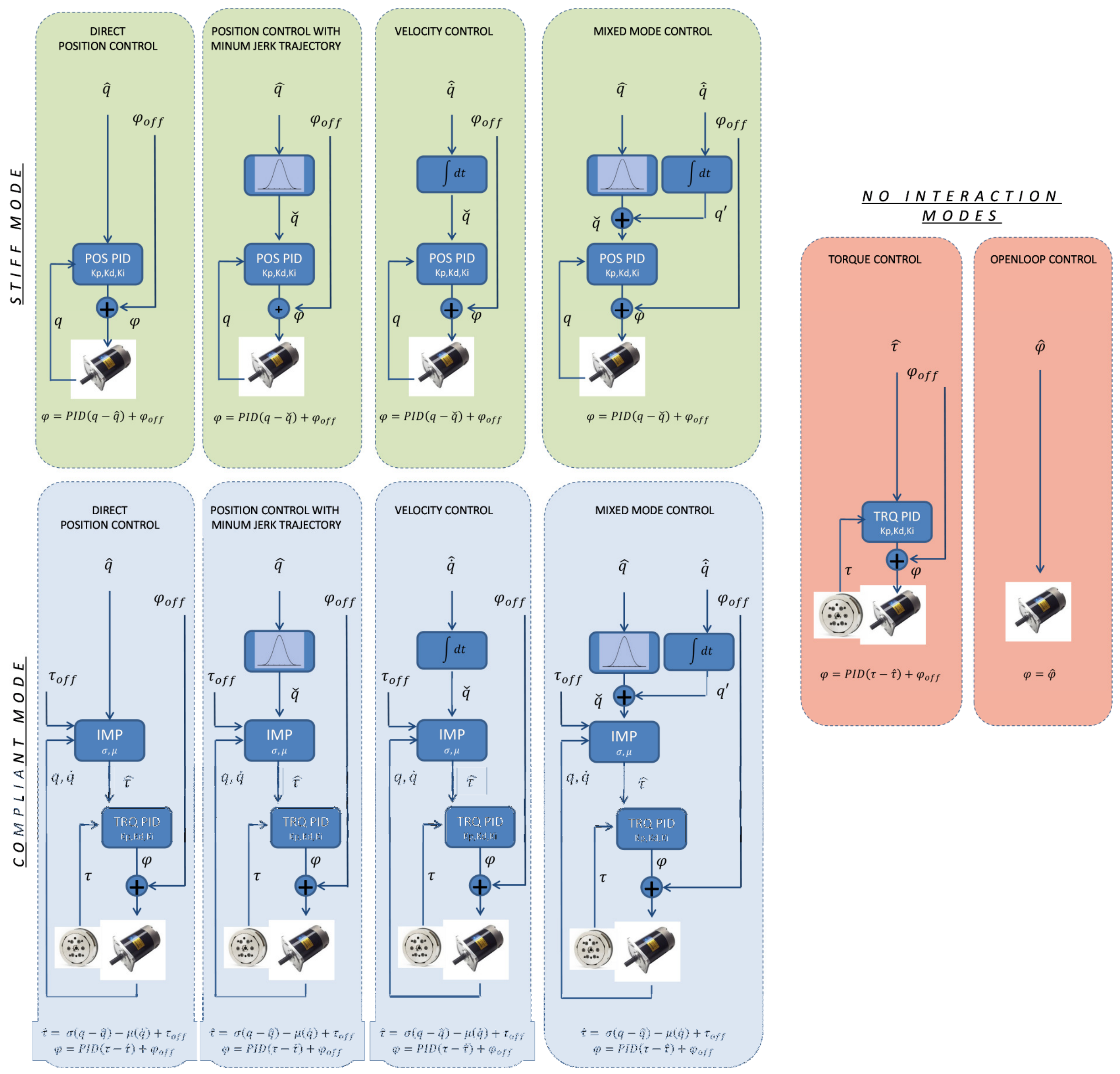
STIFF MODE
COMPLIANT MODE
NO INTERACTION MODES
DIRECT POSITION CONTROL
POSITION CONTROL WITH MINUM JERK TRAJECTORY
VELOCITY CONTROL
MIXED MODE CONTROL
TORQUE CONTROL
OPENLOOP CONTROL
POS PID Kp,Kd,Ki
IMP σ,μ
TRQ PID Kp,Kd,Ki
$\int dt$
$\varphi = PID(q - \hat{q}) + \varphi_{off}$
$\varphi = PID(q - \check{q}) + \varphi_{off}$
$\hat{\tau} = \sigma(q - \hat{q}) - \mu(\dot{q}) + \tau_{off}$
$\varphi = PID(\tau - \hat{\tau}) + \varphi_{off}$
$\hat{\tau} = \sigma(q - \check{q}) - \mu(\dot{q}) + \tau_{off}$
$\varphi = \hat{\varphi}$

Figure 2: Control algorithm schemes

3. Additional Control Board statuses

In addition to the control modes described in the previous section, there are supplementary modes (shown in Table 4) used to represent special statuses of the board and supplementary commands (shown in Table 5) used to send special commands to the board.

10. Hardware fault	VOCAB_CM_HW_FAULT
11. Calibration in progress	VOCAB_CM_CALIBRATING
12. Not Configured	VOCAB_CM_NOT_CONFIGURED

Table 4: Special board statuses

- These values cannot be set by the user with a call to the method yarp::dev::IControlMode:: setControlMode().
- These values are a valid answers for the method yarp::dev::IControlMode::getControlMode()

13. Hardware fault	VOCAB_CM_FORCE_IDLE

Table 5: Special board commands

- This command can be sent to the board using the yarp::dev::IControlMode::setControlMode() method.
- This value is never returned by yarp::dev::IControlMode::getControlMode()

3.1.'Hardware fault' status

The "hardware fault" status is used to signal both an hardware fault (e.g. broken encoder, overcurrent ecc.), or a user fault (e.g. trying to set a control mode which is not implemented by the control board).

- When the board is in "hardware fault", the controller is off (no PWM is given to the motor).
- If a control board is faulted, the user has to send the special command setControlMode(VOCAB_CM_FORCE_IDLE) to reset the fault before choosing any other desired control mode (Figure 3). Requests to switch to other modalities (including VOCAB_CM_IDLE) are ignored. The purpose is to prevent any automatic switch (e.g. from a user software module) from the fault status to an active control status.
- If the hardware fault is successfully cleared by the VOCAB_CM_FORCE_IDLE command, getControlMode() will return a standard VOCAB_CM_IDLE. If the fault persists (e.g. a critical sensor is broken) the board will remain in the fault status. Note that VOCAB_CM_FORCE_IDLE is never returned by the method getControlMode().
- Sending the VOCAB_CM_FORCE_IDLE special command when the board is not in fault status has the same effect of setting VOCAB_CM_IDLE control mode.

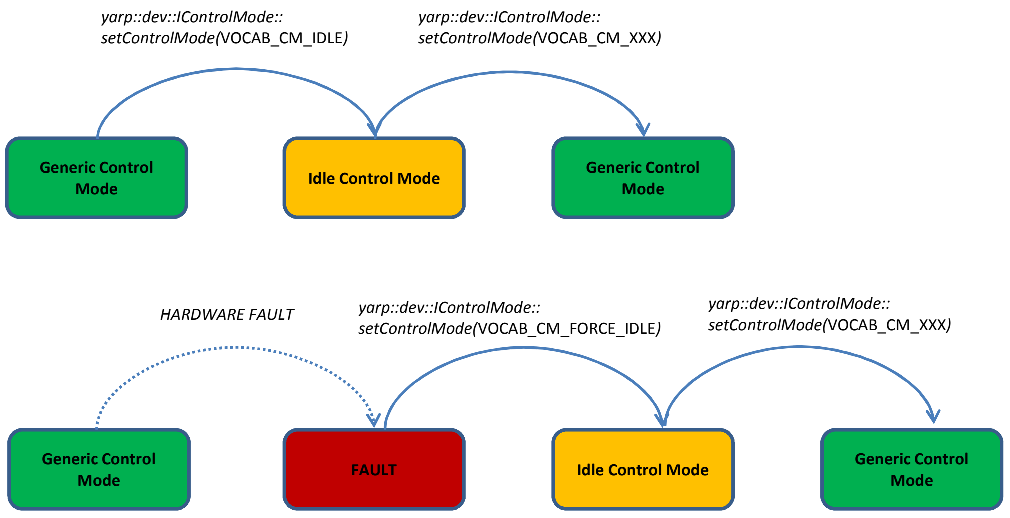
yarp::dev::IControlMode::
setControlMode(VOCAB_CM_IDLE)
yarp::dev::IControlMode::
setControlMode(VOCAB_CM_XXX)
Generic Control Mode
Idle Control Mode
Generic Control Mode
HARDWARE FAULT
yarp::dev::IControlMode::
setControlMode(VOCAB_CM_FORCE_IDLE)
yarp::dev::IControlMode::
setControlMode(VOCAB_CM_XXX)
Generic Control Mode
FAULT
Idle Control Mode
Generic Control Mode

Figure 3: FAULT status and Idle Control Mode

3.2. 'Calibration in progress' status

A board which is currently performing the calibration procedure internally sets its control mode to the special status "calibration in progress" .

- In this special status the controller is on (PWM is given to the motor). The control algorithm depends on the specific type of calibrator (e.g. absolute encoder, movement to the hardware limit etc). This is not discussed in this document.
- In this special status the interaction mode is always ignored.

Transition rules:

- The calibration procedure is requested during the initialization phase. See Section 3.3
- The calibration procedure is requested by the user during normal operation. In this case:
    1. The joint status is set to calibration in progress.
    2. The calibration algorithm is executed.
    3. After the calibration, the control mode and interaction mode is set to a value decided by the calibrator type. Default value on the iCub platform is:
        - ☑ position control, stiff interaction mode
        - ☐ position control, compliant interaction mode (if supported by the hardware)
- If the calibration fails, the board sets its internal status to 'hardware fault'.

3.3.'Not Configured' status

A board which has been just turned on internally sets its control mode to the special status "not configured".

- In this special status the controller is off (no PWM is given to the motor).
- In this special status the interaction mode is always ignored.

The following configuration values must be sent to the board in order to complete the configuration phase:

- All the controller parameters (PID gains, SW joints limits, max output, max current etc.)
- All the calibration parameters (encoders zeroes, etc.) [ONLY FOR JOINTS REQUIRING CALIBRATION]

Transition rules:

- If a joint requires calibration, then the joint exists from “not configured” status when the calibration command is received. The control mode is thus set to “calibration in progress”.
- If a joint does not require calibration, then the joint exists from “not configured status” when all the controller parameters are received. The control mode is thus set to the default value here recommended:
  - ☑ Idle status
  - ☐ position control, stiff interaction mode
  - ☐ position control, compliant interaction mode (if supported by the hardware)

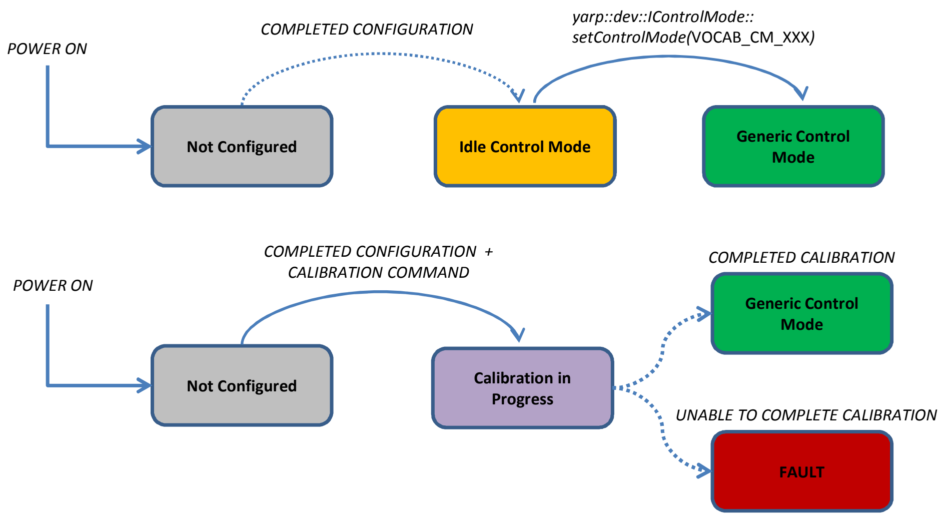

Figure 4: Transitions from a ‘Not configured’ status to valid control mode status

4. APIs

In this section the main motion control interfaces of yarp::dev are shown. Please note that not all the interfaces methods are shown here, but only the ones which allow to send a motion command to the joint. For a complete reference, please visit: http://wiki.icub.org/yarpdoc/namespaceyarp_1_1dev.html

- *yarp::dev::IPositionControl*

```
o  positionMove (int j, double ref)
o  setRefSpeed (int j, double sp)
o  setPositionMode () = 0 [pure virtual]
```

This interface is used to send a trajectory command to a joint set in either Position Mode or Mixed Mode. SetPositionMode() is a method which can optionally implemented as shortcut to set one or more joints in Position trajectory mode without acquiring the yarp::dev::IControlModeInterface.

- ***yarp::dev::IPositionDirect***
  - `setPosition (int j, double ref)`
  - `setPositionDirectMode () = 0 [pure virtual]`

This interface is used to send direct position (step) commands to a joint. The joint should be only in Position direct mode. SetPositionDirectMode() is a method which can optionally implemented as shortcut to set one or more joints in Position direct mode without acquiring the yarp::dev::IControlModeInterface.

- ***yarp::dev::ITorqueControl***
  - `setRefTorque (int j, double t)`
  - `setTorqueMode () = 0 [pure virtual]`

This interface is used to send torque commands to a joint. The joint must be in Torque mode.
SetTorqueMode() is a method which can optionally implemented as shortcut to set one or more joints in Torque mode without acquiring the yarp::dev::IControlModeInterface.

- ***yarp::dev::IVelocityControl***
  - `velocityMove (int j, double sp)`
  - `setVelocityMode () = 0 [pure virtual]`

This interface is used to send velocity commands to a joint set in either Velocity Mode or Mixed Mode.
SetVelocityMode() is a method which can optionally implemented as shortcut to set one or more joints in Velocity mode without acquiring the yarp::dev::IControlModeInterface.

- ***yarp::dev::IOpenLoopControl***
  - `setOutput (int j, double t)`
  - `setOpenLoopMode () = 0 [pure virtual]`

This interface is used to send openloop commands (e.g. PWM control) to a joint set in OpenLoop Mode.
SetOpenloopMode() is a method which can optionally implemented as shortcut to set one or more joints in OpenLoop mode without acquiring the yarp::dev::IControlModeInterface.

- ***yarp::dev::IControlMode2***
  - `bool getControlMode (int j, int* mode)`
  - `bool getControlMode (int* modes)`
  - `bool setControlMode (int j, const int mode)`
  - `bool setControlMode (const int* modes)`
  - `bool setControlMode (const int n_joint, const int *joints, int *modes)`

```
#define VOCAB_CM_POSITION         VOCAB3('p','o','s'),
#define VOCAB_CM_POSITION_DIRECT  VOCAB4('p','o','s','d'),
#define VOCAB_CM_VELOCITY         VOCAB4('v','e','l'),
#define VOCAB_CM_MIXED            VOCAB3('m','i','x'),
#define VOCAB_CM_TORQUE           VOCAB4('t','o','r','q'),
#define VOCAB_CM_OPENLOOP         VOCAB4('o','p','e','n'),
#define VOCAB_CM_IDLE             VOCAB3('i','d','l'),
#define VOCAB_CM_FORCE_IDLE       VOCAB4('f','i','d','l'),
#define VOCAB_CM_HW_FAULT         VOCAB4('h','w','f','a'),
#define VOCAB_CM_CALIBRATING      VOCAB3('c','a','l'),
#define VOCAB_CM_CALIB_DONE       VOCAB4('c','a','l','d'),
#define VOCAB_CM_NOT_CONFIGURED  VOCAB4('c','f','g','n'),
#define VOCAB_CM_CONFIGURED      VOCAB4('c','f','g','y')
```

This interface is used to set/get the control mode of one or more joints. If the requested control mode is not implemented, setControlMode() returns false.

- ***yarp::dev::IInteractionModeEnum***

```
namespace yarp {namespace dev
{
    enum IInteractionModeEnum
    {
            VOCAB_IM_STIFF           = VOCAB4('s','t','i','f'),
            VOCAB_IM_COMPLIANT       = VOCAB4('c','o','m','p')
    };
}
```

- ***yarp::dev:IInteractionlMode***
  - `bool getInteractionMode (int j, yarp::dev::InteractionModeEnum* mode)`
  - `bool getInteractionMode (yarp::dev::InteractionModeEnum* modes)`
  - `bool getInteractionMode (int n_joints, int *joints, yarp::dev::InteractionModeEnum* modes)`
  - `bool setInteractionMode (int j, const yarp::dev::InteractionModeEnum mode)`
  - `bool setInteractionMode (const yarp::dev::interactionModeEnum* modes)`
  - `bool setInteractionMode (int n_joints, int *joints, const yarp::dev::interactionModeEnum* modes)`

This interface is used to set/get the interaction mode of one or more joints. If the requested interaction mode is not implemented, setInteractionMode() returns false.

- ***yarp::dev::IImpedanceControl***
  - `getImpedance (int j, double *stiffness, double *damping)`
  - `setImpedance (int j, double stiffness, double damping)`

This interface allows to set/get the impedance parameters that are use when the joint is set in compliant mode. This interface does not have any method to send motor command to the joints.

5. Differences with the Old APIs

In this section the key differences between the new and old interfaces are discussed. The methods of the OLD yarp::dev::IControlMode interface (now replaced by yarp::dev::IControlMode2) are here reported for reference purpose.

yarp::dev::IControlMode

- `getControlMode(int j, int *mode)=0`
- `getControlModes(int *modes)=0`
- `setImpedancePositionMode(int j)=0`
- `setImpedanceVelocityMode(int j)=0`
- `setOpenLoopMode(int j)=0`
- `setPositionMode(int j)=0`
- `setTorqueMode(int j)=0`
- `setVelocityMode(int j)=0`

- Previous APIs use a single interface, iControlMode, to set both the control mode and the interaction mode. For example, setImpedancePositionMode() method correspond to setControlMode() and setInteractionMode() (using the parameters VOCAB_CM_POSITION and VOCAB_IM_COMPLIANT respectively). This methods are currently kept for for back-compatibility (showing a “deprecated” warning message). They will removed in the near future.
- Previous APIs allow automatic switching between Position and Velocity mode (i.e. a VelocityMove () command sent in positionMode automatically sets the joint in velocityMode). This is no more supported, a velocityMove() commands are rejected, unless the MixedMode is used. Same considerations apply to the ImpedancePositionMode and ImpedanceVelocityMode.
- Previous APIs do not allow to set IdleMode (iAmplifierControl::disableAmp() method is used), but getControlMode() can return “idle” if a fault occurs on the board.
- Previous API do not distinguish between positionMode and positionDirectMode in terms of control mode (both of them are considered “position mode”).
- Previous APIs do not distinguish between idle mode and hardware fault. Now the user is forced to send a special command to reset the fault.
- The use of the following old methods should be avoided in new applications because they introduce ambiguities and are architecture-dependent. Use yarp::dev::IControlMode instead.
    - yarp::dev::IAmplifierControl::enableAmp()
    - yarp::dev::IAmplifierControl::disableAmp()
    - yarp::dev::IPidControl::enablePid()
    - yarp::dev::IPidControl::disablePid()
    - yarp::dev::iTorqueControl::enableTorquePid()
    - yarp::dev::ITorqueControl::disableTorquePid()