


electric	fuerte	groovy	hydro	indigo	jade	kinetic	lunar
----------	--------	--------	-------	--------	------	---------	-------

### Contents

1. ROS-Industrial Robot Driver Specification (Final)
  1. Motivation
  2. Operation
    1. Initialization
    2. Communications
  3. ROS API
    1. General
    2. State Feedback
    3. Motion Control
    4. Kinematics
    5. Path Accuracy
  4. Implementation
    1. Essential Capabilities
    2. Reference Implementation
    3. Hardware Compatibility List
  5. Future Plans

# 1. ROS-Industrial Robot Driver Specification (Final)

Provides guidelines for ROS node functionality, to improve cross-platform compatibility. All ROS nodes that provide an interface to an industrial robot controller should adhere to the specification.

Comments/Questions on this specification can be sent to the ROS-Industrial  mailing list (<mailto:swri-ros-pkg-dev@googlegroups.com>).

## 1.1 Motivation

One of the primary benefits of ROS-Industrial is that it enables interoperability between robots from different vendors, by integrating their control with the common ROS framework. Even though ROS integration is a good start, better compatibility is achieved when these different ROS nodes all utilize a common set of interfaces for control and feedback.

This page attempts to provide some guidelines on what specific ROS interfaces should be provided by a ROS-Industrial robot to ensure maximum compatibility. It is expected that vendor-specific implementation details may require compromises in some areas, but this will hopefully provide a “gold standard” implementation description. See below for details on a Reference Implementation (/industrial\_robot\_client) following these guidelines and a Hardware Compatibility List (/Industrial/Industrial\_Robot\_Driver\_Spec/HardwareCompatibility) that provides platform-specific information.

## 1.2 Operation

This section describes how the robot is expected to respond to high-level operational activities (e.g. startup, shutdown, etc.).

### 1.2.1 Initialization

- the ROS node should automatically initialize all connections to the robot controller
  - *no manual connect service-call should be required*
- it is preferred to have any robot-side code automatically running on controller startup
  - e.g. in a background, always-on task
  - any startup requirements should be explicitly stated in the node's documentation

### 1.2.2 Communications

- both sides of the robot-ROS connection should handle communications-loss scenarios:
  - the **ROS node** shall...
    - automatically try to reconnect to the robot (at ~1Hz intervals)
    - stop publishing most messages (e.g. position feedback)
    - continue publishing state messages, with `connected=false`
  - the **robot** shall...
    - stop motion and power off the drives
    - reinitialize all communications to wait for a new connection
  - if the interface cannot detect comms-loss directly, it may be necessary to implement a heartbeat message between the robot/ROS sides.
- no mechanism is specified to flag something as **unsupported**
  - topics should not be published (or subscribed to) if unsupported
  - if a service call is advertised by a node, it is assumed to be supported (i.e. return valid results)
    - *standard ROS mechanisms will ensure the client gets an error if it calls an unimplemented service*

## 1.3 ROS API

This section describes specific ROS interfaces (topics, services, and parameters) that the robot should provide. These capabilities may be provided by a single node or multiple nodes, as needed by the robot communications architecture.

It is also possible that different nodes may be used to provide the same capabilities with different assumptions, communications methods, etc. ROS launch files can be used to control which nodes are appropriate to run for a given application.

### 1.3.1 General




#### Parameters

- `robot_ip_address` (string)
  - IP address of robot connection
- `robot_description` (urdf map)

- The urdf (/urdf) xml robot description

## 1.3.2 State Feedback

### Published Topics

- `feedback_states` (  `control_msgs/FollowJointTrajectoryFeedback`  
([http://docs.ros.org/groovy/api/control\\_msgs/html/action/FollowJointTrajectory.html](http://docs.ros.org/groovy/api/control_msgs/html/action/FollowJointTrajectory.html))-lowest part)
  - provide feedback of current vs. desired joint position (and velocity/acceleration)
  - used by ROS-I's  `joint_trajectory_action` ([https://github.com/ros-industrial/industrial\\_core/blob/groovy-devel/industrial\\_robot\\_client/src/joint\\_trajectory\\_action.cpp](https://github.com/ros-industrial/industrial_core/blob/groovy-devel/industrial_robot_client/src/joint_trajectory_action.cpp)) to monitor in-progress motions
    - joint names, ordering, and scale should match ROS conventions ("e.g. joints ordered from base-to-tool, angles in radians, etc.")
    - if desired and error values unavailable, leave arrays **empty**
    - if velocities and accelerations unavailable, leave arrays **empty**
- `joint_states` (`sensor_msgs/JointState`  
([http://docs.ros.org/api/sensor\\_msgs/html/msg/JointState.html](http://docs.ros.org/api/sensor_msgs/html/msg/JointState.html)))
  - provide feedback of current joint position (and velocity/effort)
  - used by the `robot_state_publisher` (/robot\_state\_publisher) node to broadcast kinematic transforms
    - joint names, ordering, and scale should match ROS conventions (\_e.g. radians\_)
    - if velocity and effort unavailable, leave arrays **empty**
- `robot_status` (  `industrial_msgs/RobotStatus`  
([http://docs.ros.org/groovy/api/industrial\\_msgs/html/msg/RobotStatus.html](http://docs.ros.org/groovy/api/industrial_msgs/html/msg/RobotStatus.html)))
  - provide current status of critical robot parameters
  - used by application code to monitor and react to different fault conditions
    - we prefer using explicit message-fields (instead of Diagnostics (/diagnostics) package) for better client-code usability
    - status values should be set to -1 if not yet available/implemented

## 1.3.3 Motion Control

This node implements methods to control the robot's movement.

Several different types of control may be implemented (trajectory download, streaming points, force-control, etc.), and may not all be compatible with each other. The robot implementation can provide these methods in a single "master motion node", or as discrete single-purpose nodes. *It is up to the implementation to ensure that conflicting input from different motion controllers is handled properly.* in general, only one controller should be commanding robot motion at a time. For complex systems, it may be appropriate to have a "motion manager" node that aggregates and prioritizes competing commands coming from different motion controllers.

### Subscribed Topics



- `joint_path_command` (`trajectory_msgs/JointTrajectory`  
([http://docs.ros.org/api/trajectory\\_msgs/html/msg/JointTrajectory.html](http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectory.html)))

- execute a pre-calculated joint trajectory on the robot
- used by ROS's trajectory generators (e.g. joint\_trajectory\_action (/joint\_trajectory\_action)) to issue motion commands
  - often, the entire trajectory is downloaded to the robot, then executed
    - *if necessary, download the trajectory point-by-point, then execute all at once*
    - alternatively, this could stream the individual path points to the JointCommand interface (*see below*)
  - if the trajectory size exceeds the maximum size (*robot-specific*), the node should ignore the trajectory and log an error
    - *it may be necessary for calling code to use a downsampling filter before this node, to reduce the path size*
  - the robot should attempt to automatically start motion, if possible
    - *this may require enabling drives, executing a motion task, etc.*
  - motion should stop (but drives stay on) when the trajectory is completed
  - if a new trajectory is received while an existing motion is in-progress:
    1. cancel the current motion (*robot motion may stop*)
    2. start the new trajectory
    3. all trajectory points are executed in-sequence, using the specified velocities.  
*NOTE: if a non-deterministic protocol (e.g. UDP) is used for transmitting the trajectory to the robot, some method should be used to ensure that the correct sequence is maintained*
  - The time\_from\_start may be used instead of velocity information to control the trajectory. This assumes that 'time\_from\_start' values were calculated for continuous motion (which is general true, but not always for ROS).
- joint\_command (trajectory\_msgs/JointTrajectoryPoint  
[http://docs.ros.org/api/trajectory\\_msgs/html/msg/JointTrajectoryPoint.html](http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectoryPoint.html))
  - execute dynamic motion by streaming joint commands on-the-fly
  - used by client code to control robot position in "real-time"
    - the robot implementation may use a small buffer to allow smooth motion between successive points
    - the topic publisher is responsible for providing new commands at sufficient rate to keep up with robot motion.
      - *denser paths (or faster robot motion) will require higher update rates*
    - this node should monitor the topic stream to ensure points are not received out-of-order (e.g. using time\_from\_start).
      - commands from the past should be dropped.
    - if commands are received faster than the robot can process, behavior may be implementation-dependent:
      1. **preferred:** immediately cancel current motion and begin move to new point. As long as motion remains smooth.


2. **alternative**: new commands will replace the previous most-recent command on the robot-side command queue.

## Services

*All services should return a value, so the client knows if the call succeeded or failed.*

- `stop_motion` (  `industrial_msgs/StopMotion`  
([http://ros.org/doc/groovy/api/industrial\\_msgs/html/srv/StopMotion.html](http://ros.org/doc/groovy/api/industrial_msgs/html/srv/StopMotion.html)))
  - stop current robot motion
  - can resume motion by sending a new motion command
- `joint_path_command` (  `industrial_msgs/CmdJointTrajectory`  
([http://ros.org/doc/api/industrial\\_msgs/html/srv/CmdJointTrajectory.html](http://ros.org/doc/api/industrial_msgs/html/srv/CmdJointTrajectory.html)))
  - execute a new motion trajectory on the robot
  - identical functionality to the `joint_path_command` topic
  - the service-implementation allows calling code to receive confirmation that a commanded trajectory has actually been received by a robot node.

### 1.3.4 Kinematics

To enable real-time motion planning and collision avoidance, the node should provide robot-specific inverse kinematics solutions. A generic solver is provided in ROS, but it operates too slowly for collision-avoidance path planning. As described here ([/arm\\_navigation/Tutorials/Running%20arm%20navigation%20on%20non-PR2%20arm](#)) and here ([/Industrial/Tutorials/Create\\_a\\_Fast\\_IK\\_Solution](#)), the  `ikfast` ([http://openrave.org/docs/latest\\_stable/openravepy/ikfast/](http://openrave.org/docs/latest_stable/openravepy/ikfast/)) tool can be used to generate the required kinematics files for any 6-DOF serial manipulator. These files should be integrated with ROS as a **plugin**, not a **service**, to avoid expensive communications-related overhead. This involves creating a launch file to connect your plugin with the ROS `arm_kinematics_constraint_aware` node, as described here ([/arm\\_kinematics\\_constraint\\_aware/Tutorials/Configuring%20kinematics%20for%20your%20arm](#)).

### 1.3.5 Path Accuracy

The motion interfaces outlined above do not specify the required accuracy for following a particular trajectory. The level of path accuracy achieved will depend on limitations of the specific robot, controller, and ROS-interface driver.

#### Collision Padding

The ROS path planners and collision checkers use high-order smoothing of trajectories between waypoints. The resulting trajectory, as executed by the robot controller, will follow a similar “smooth” trajectory, “but may not exactly match the “ideal” planned trajectory”. For this reason, the ROS path planners add a specified amount of “padding” to the robot models to account for differences between the planned and actual paths. This results in a very high probability of collision-free motion. Increasing the level of padding reduces the chance of collisions.

A closer integration between path planners and robot controllers would reduce both the path-execution error and the collision-padding requirement. However, this level of integration is beyond the scope of current ROS-Industrial efforts and may require robot-specific solutions.

## 1.4 Implementation

This section provides details on current implementations of ROS-Industrial, and how closely they follow these guidelines.

### 1.4.1 Essential Capabilities

The following capabilities are essential for robots to interact smoothly with other ROS-Industrial components:

- publish `feedback_states`
  - *at a minimum, the names and positions fields*
- publish `joint_states`
  - *at a minimum, the names and actual \positions fields*
- subscribe to `JointPathCommand`
  - *at a minimum, execute the joint-positions in sequence at a default velocity*

### 1.4.2 Reference Implementation

A reference implementation of a ROS-Industrial node implementing these capabilities can be found here ([/industrial\\_robot\\_client](#)). This implementation uses a simple message-based socket protocol to communicate with industrial robots.

New robot implementations are encouraged to consider using this client, if appropriate, for their robot. See the ABB and Motoman implementations for examples of how to integrate a robot-side server application with a ROS-Industrial client node derived from this reference implementation.

### 1.4.3 Hardware Compatibility List

The Hardware Compatibility List ([/Industrial/Industrial\\_Robot\\_Driver\\_Spec/HardwareCompatibility](#)) documents which features outlined in this page are supported by various robot-specific implementations. Consult this list before developing your application, to determine whether your target robot platform supports all the necessary ROS interfaces.

## 1.5 Future Plans

The Future Plans ([/Industrial/Industrial\\_Robot\\_Driver\\_Spec/future](#)) page lists a few ideas we have for expanding the functionality of ROS-Industrial nodes. These items still need some refinement/discussion, so are not specified above. We welcome your suggestions on any of these items. Or, feel free to suggest more features!

Except where otherwise noted, the

ROS wiki is licensed under the

Wiki: Industrial/Industrial\_Robot\_Driver\_Spec (last edited 2015-05-08 15:03:51 by GvdHoon (/GvdHoon))

Creative Commons Attribution 3.0

(<http://creativecommons.org/licenses/by/3.0/>) | Find us on Google+

(<https://plus.google.com/113789706402978299308>)

Brought to you by:  Open Source Robotics Foundation

(<http://www.osrfoundation.org>)