RobCo Profinet User Manual

Integration into SIEMENS TIA Portal

RobCo GmbH

2026-10-30

Contents

1 Introduction **3**

2 Connect PROFINET to RobCo Robot **3**

3 TIA Portal Application Manual **4**

3.1 Overview . . . 4
3.2 TIA Portal Setup . . . 4
3.2.1 Handover files . . . 4
3.2.2 Demo project . . . 5
3.2.3 Install and add the device . . . 5
3.2.4 Assign device name and IP . . . 5
3.3 Configure the Device . . . 5
3.3.1 Table: Module and Submodule Layout . . . 5
3.3.2 Import PLC data types . . . 6
3.3.3 Create PLC tags . . . 6
3.4 Process Image Reference . . . 7
3.4.1 Table: Core Input Relative Offsets . . . 7
3.4.2 Table: Core Output Relative Offsets . . . 9
3.4.3 Table: Optional Base Submodule Relative Offsets . . . 9
3.5 Command Handling . . . 11
3.5.1 Command handshake . . . 11
3.5.2 SET_BRAKES example . . . 12
3.6 Single Point of Control (SPOC) . . . 12
3.6.1 Authority status . . . 12
3.6.2 PLC takes control: CLAIM_CONTROL . . . 12
3.6.3 Return control to a browser . . . 13
3.6.4 Motion and streaming during a transfer . . . 13
3.6.5 Backend restart . . . 14
3.7 Command Reference . . . 14
3.7.1 Table: cmd_id Values . . . 14
3.7.2 Table: last_req_status Values . . . 15
3.7.3 Acknowledge an error . . . 15
3.7.4 Legacy operation-mode byte . . . 15
3.8 AuxVariables . . . 15
3.8.1 Table: AuxVariable Directional Bank Mapping . . . 16
3.8.2 Table: AuxVariable Types . . . 16

3.8.3 Typed overlay 17
3.9 Commissioning and Troubleshooting 17
3.9.1 Commissioning sequence 17
3.9.2 Troubleshooting 18
3.10 Value Tables 19
3.10.1 Table: flow_state Values 19
3.10.2 Table: robot_state Values 20
3.10.3 Table: safety_state Values 21

1 Introduction

This handbook provides a comprehensive overview of the PROFINET IO-Device implementation developed by RobCo GmbH, intended for the integration with Siemens TIA Portal.

Vendor ID
Name: RobCo GmbH
DECIMAL: 1950
HEX: 0x079E

Profinet Protocol

- Profinet v2.45
  - Conformance Class A and B
  - Real Time Class 1
  - pure IO-Device (no Controller functionality)

2 Connect PROFINET to RobCo Robot

Connect the `peripheral port` of the **RobCo Control Unit** to the industrial network.

3 TIA Portal Application Manual

Last change: 2026-10-30

This manual describes how to engineer and program the RobCo Modular Robot PROFINET v2 device in Siemens TIA Portal. It contains the complete PLC-facing layout, command interface and published state values.

Tested environments:

- Siemens TIA Portal V19
- Siemens S7-1200 and S7-1500 PLCs.

3.1 Overview

PROFINET directions always use the controller perspective:

- **Input** or `%I`: RobCo device -> PLC.
- **Output** or %Q: PLC -> RobCo device.

RobFlow uses the application perspective for AuxVariables: `profinetOutput` maps to a PROFINET Input module, while `profinetInput` maps to a PROFINET Output module.

PROFINET v2 is a breaking change from the old flat process image. Do not reuse old `RobCo_Input` or `RobCo_Output` tags. For an existing TIA project, remove the old RobCo device and add the v2 device again unless TIA demonstrably recreates every module and address correctly.

3.2 TIA Portal Setup

3.2.1 Handover files

Keep these files together:

- `GSDML-V2.45-RobCo_GmbH-ModularRobot-20260901.xml`
- `GSDML-079e-0001-Favicon.bmp`
- `RobCo_Datatypes.udt`
- `RobCo_Profinet_Variables.udt`

`RobCo_Datatypes.udt` defines the fixed process image. The second file is a project-specific typed AuxVariable overlay and is replaced by the RobFlow export whenever the configured variables change.

Use the type names from the supplied `RobCo_Datatypes.udt`, as listed in Create PLC tags. Older demo projects used the names `RobCo_Profinet_CartesianPose` and `RobCo_Profinet_JointPose`; update those project types and their references to match the supplied definitions before importing an AuxVariable overlay. A type-name change does not change the wire layout.

3.2.2 Demo project

`demo_projects/Profinet-v2_hmi.zap19` contains a PROFINET v2 demo project with HMI for TIA Portal V19. The PLC release ZIP includes it under `tia/`. Retrieve the archive in TIA Portal and adapt the device names, network addresses and AuxVariable mappings to the target installation.

3.2.3 Install and add the device

1. Select `Options` -> `Manage general station description files (GSD)`.
2. Select the directory containing the XML and BMP files and install the GSDML.
3. Add `RobCo Modular Robot` from `Other field devices` -> `PROFINET IO` -> `I/O` -> `RobCo GmbH` -> `RobCo Devices`.
4. Connect the device PN/IE port to the PLC IO system.

Do not use the RobCo-internal networks `192.168.3.0/24` or `192.168.4.0/24` for the customer PROFINET network.

3.2.4 Assign device name and IP

1. Open the RobCo device under the PLC PROFINET IO system.
2. Select `Online & diagnostics` -> `Functions` -> `Assign PROFINET device name`.
3. Refresh accessible devices and use the signal LED for identification.
4. Assign the station name configured in the TIA project.

The IO controller assigns the configured IP address by DCP.

3.3 Configure the Device

3.3.1 Table: Module and Submodule Layout

Module / submodule	Location	Input bytes	Output bytes	Use
Core Data	slot 1/subslot 1	92	84	mandatory
Payload IO Submodule	slot 1/subslot 2	40	40	optional; `SET_PAYLOAD`
CartesianPose IO Submodule	slot 1/subslot 3	24	24	optional; `MOVE_CARTESIAN`
JointPose IO Submodule	slot 1/subslot 4	32	32	optional; `MOVE_JOINT`
Aux Input 64B	slot 2..13/subslot 1	64	0	optional cyclic data
Aux Output 64B	slot 2..13/subslot 1	0	64	optional cyclic data

Slots 2 through 13 each accept one Aux Input or Aux Output module. Sparse slots are valid.

3.3.2 Import PLC data types

1. Add `RobCo_Datatypes.udt` under `External source files`.
2. Add `RobCo_Profinet_Variables.udt` when AuxVariables are used.
3. Select `Generate blocks from source` for each file.

Payload, CartesianPose, JointPose and Aux64 use the same neutral UDT in both directions because their binary layouts are identical. Core Data uses separate types because its Input and Output layouts differ.

3.3.3 Create PLC tags

TIA assigns a separate absolute start address to every submodule direction. The programmer may change these addresses. Always place each tag at the actual start shown in Device view; addresses in this manual are relative to that tag.

Example tag	UDT
`robco_core_in`	`RobCo_Core_Input`

Example tag	UDT
`robco_core_out`	`RobCo_Core_Output`
`robco_payload_in`, `robco_payload_out`	`RobCo_Payload`
`robco_cartesian_pose_in`, `robco_cartesian_pose_out`	`RobCo_CartesianPose`
`robco_joint_pose_in`, `robco_joint_pose_out`	`RobCo_JointPose`
`robco_aux_in`, `robco_aux_out`	exported overlay or `RobCo_Aux64`

Only create tags for plugged submodules and supplied directions.

3.4 Process Image Reference

All offsets are relative to the separately addressed submodule. Multi-byte numbers use big-endian encoding; `Real` uses IEEE-754. UUID fields are fixed `Char` arrays containing the canonical ASCII UUID; an all-zero field represents the NIL UUID. AuxVariable strings use the encoding in AuxVariable Types.

3.4.1 Table: Core Input Relative Offsets

Relative byte	UDT field and S7 type	Meaning
0..35	`current_flow_uuid: Array[0..35] of Char`	current flow UUID
36..71	`current_tool_uuid: Array[0..35] of Char`	current tool UUID
72..75	`current_global_speed: UDInt`	percentage
76..77	`last_error_code: UInt`	latest backend error code
78..79	`flow_state: UInt`	flow_state
80..81	`robot_state: UInt`	robot_state
82..83	`safety_state: UInt`	safety_state
84..85	`status.ack_seq: UInt`	accepted sequence
86..87	`status.last_req_seq: UInt`	result sequence

Relative byte	UDT field and S7 type	Meaning
88..89	`status.last_req_status: UInt`	request status

All named Core Input flags are `Bool` values.

Relative bit	UDT field
90.0	`flags.operational`
90.1	`flags.robot_in_home_position`
90.2	`flags.teach_mode`
90.3	`flags.safety_auto_mode`
90.4	`flags.safety_manual_mode`
90.5	`flags.flow_running`
90.6	`flags.flow_paused`
90.7	`flags.flow_ready_to_start`
91.0	`flags.brakes_closed`
91.1	`flags.system_alarm_error`
91.2	`flags.emergency_stop_triggered`
91.3	`flags.hand_guiding`
91.4	`flags.control_external_active`
91.5	`flags.control_local_active`
91.6	`flags.enabling_device_pressed`
91.7	reserved

The authority flags report the selected Single Point of Control owner; see Single Point of Control. `enabling_device_pressed` mirrors the physical SCU enabling switch. It is false when detailed SCU status is unavailable or invalid. It does not grant control authority. `emergency_stop_triggered` indicates a safe-stop state, including safe stops other than a physical emergency stop. These cyclic status flags are diagnostic information, not safety-rated signals.

3.4.2 Table: Core Output Relative Offsets

Relative byte	UDT field and S7 type	Used by
0..35	`requested_flow_uuid`: `Array[0..35] of Char`	`START_FLOW`
36..71	`requested_tool_uuid`: `Array[0..35] of Char`	`SET_TOOL`
72..75	`requested_global_speed`: `UDInt`	`SET_GLOBAL_SPEED`
76..77	`command.cmd_id`: `UInt`	cmd_id
78..79	`command.cmd_seq`: `UInt`	command sequence
80..81	`flags`: two-byte bit field	command flags
82	`requested_operation_mode`: `USInt`	legacy field name; unused, write zero
83	`pad_byte`: `Byte`	always write zero

Relative bit	UDT field	Used by
80.0	`flags.brakes_closed`	`SET_BRAKES`
80.1	`flags.acknowledge_error`	legacy field; unused, write false
80.2	`flags.hand_guiding`	`SET_HAND_GUIDING`
80.3..81.7	reserved	write zero

`pad_byte` is explicit protocol data that keeps the GSDML and imported UDT at 84 bytes; it is not an implicit PROFINET alignment byte.

3.4.3 Table: Optional Base Submodule Relative Offsets

Each table starts at byte zero of its separately addressed submodule. Input contains current values; Output contains requested values.

3.4.3.1 Payload IO Submodule: current / requested payload Both directions use `RobCo_Payload`.

Relative byte	UDT field	S7 type
0	mass	Real
4	com_x	Real
8	com_y	Real
12	com_z	Real
16	ixx	Real
20	ixy	Real
24	ixz	Real
28	iyy	Real
32	iyz	Real
36	izz	Real

The inertia fields are the upper triangle of the symmetric tensor, ordered `ixx, ixy, ixz, iyy, iyz, izz`, in `kg*m^2`.

3.4.3.2 CartesianPose IO Submodule: current / requested TCP pose Both directions use `RobCo_CartesianPose`.

Relative byte	UDT field	S7 type
0	position_x	Real
4	position_y	Real
8	position_z	Real
12	orientation_rz	Real
16	orientation_ry	Real
20	orientation_rx	Real

3.4.3.3 JointPose IO Submodule: current / requested joint pose Both directions use `RobCo_JointPose`.

Relative byte	UDT field	S7 type
0	joint_angle_1	Real
4	joint_angle_2	Real
8	joint_angle_3	Real
12	joint_angle_4	Real
16	joint_angle_5	Real
20	joint_angle_6	Real
24	joint_angle_7	Real
28	joint_angle_8	Real

3.5 Command Handling

3.5.1 Command handshake

A changed cmd_seq identifies every new command. Changing payload data or cmd_id alone does not create a request.

1. Write all payload fields.
2. Write cmd_id.
3. Increment cmd_seq once. Wrap-around from 65535 to 0 is valid.
4. Wait for ack_seq == cmd_seq.
5. Wait for last_req_seq == cmd_seq.
6. Evaluate last_req_status.

Interpret the result using last_req_status Values. The PLC need not observe every intermediate status: a command can finish before the next PLC scan. After reading the final result, the PLC may write cmd_id = NONE without incrementing cmd_seq.

Keep one request outstanding at a time and hold its command and payload unchanged until the matching final result. After connecting or restarting the PLC program, read ack_seq before choosing the next sequence; a reused sequence does not start a new request. On a timeout or connection loss, check the robot and result state before deciding whether to send a new request. Do not automatically repeat a motion command.

DONE reports completion of the command handler. A started flow or movement may still be active; use the published state and feedback to follow it.

3.5.2 SET_BRAKES example

1. Write `robco_core_out.flags.brakes_closed`.
2. Write `SET_BRAKES` to `robco_core_out.command.cmd_id`, using the value in cmd_id Values.
3. Increment `robco_core_out.command.cmd_seq`.
4. Apply the command handshake above.

All commands use this mechanism.

3.6 Single Point of Control (SPOC)

RobFlow grants command authority to at most one owner: a browser session or the PROFINET PLC. Other browser sessions can observe the robot, but only the current owner may issue motion commands. Cyclic IO exchange alone does not grant the PLC command authority.

3.6.1 Authority status

The Core Input flags report the selected owner:

`control_external_active`	`control_local_active`	Selected owner
true	false	PROFINET PLC
false	true	RobFlow browser session
false	false	No owner / authority unavailable

Only issue ordinary PLC commands while `control_external_active` is true. The backend rejects them with `REJECTED_UNAUTHORIZED` otherwise, including `STOP` and `ACK_ERROR`. `CLAIM_CONTROL` is the exception: it must be callable before the PLC owns control. Ownership is independent of the robot's safety and operating states; an authority flag does not bypass motion prerequisites or safety functions.

3.6.2 PLC takes control: CLAIM_CONTROL

`CLAIM_CONTROL` transfers authority to the PLC when a browser or nobody holds it. It has no payload and requires only the mandatory Core Data submodule.

1. Write `CLAIM_CONTROL` to `robco_core_out.command.cmd_id`, using the value in cmd_id Values.
2. Increment `robco_core_out.command.cmd_seq` once.
3. Wait for the matching `ack_seq` and final `last_req_status` as described in the command handshake.
4. Require DONE and `control_external_active = true` before issuing another command. Authority status is published cyclically and may lag the result.

Taking control supersedes the browser owner and stops robot motion through `Robot.stop()`. A claim while the PLC already owns control is rejected; do not send `CLAIM_CONTROL` repeatedly from every PLC scan. Failed claims must be handled through the normal result handshake.

3.6.3 Return control to a browser

Disable PROFINET control in the RobFlow browser interface. This grants authority directly to the requesting browser and calls `Robot.stop()`. The PLC can only claim control; `CLAIM_CONTROL` cannot release it or hand it to another owner. A normal browser control request cannot displace the PLC while PROFINET control remains enabled.

3.6.4 Motion and streaming during a transfer

Authority transfer	Motion behavior
Browser / no owner -> PLC	Stop robot motion, including active streaming.
PLC -> browser	Stop robot motion, including active streaming.
Browser A -> browser B	Continue the current motion or stream; no stop solely due to this transfer.

The streaming interface is treated like a movement executed by a flow. `Robot.stop()` stops it as well as flow motion, so a transfer to PLC control terminates an active stream. A transfer does not automatically restart the stopped motion; the new owner must explicitly initiate subsequent motion. Streaming through another interface is independent of the PLC command interface; the available PLC commands are listed in cmd_id Values.

3.6.5 Backend restart

PLC ownership is persisted and restored after a backend restart or robot reboot; RobFlow reacquires its robot-control lease in the background. Browser session ownership is not restored. Restoration does not start motion. After reconnecting, recheck authority, robot state and command results before sending motion commands.

3.7 Command Reference

3.7.1 Table: cmd_id Values

`cmd_id` / command	Payload and result
0 `NONE`	No payload; no command.
1 `SET_BRAKES`	Core Output brake flag.
2 `SET_GLOBAL_SPEED`	Core Output global speed, `1..100`; invalid range returns `REJECTED_VALUE`.
3 `SET_PAYLOAD`	Payload IO Submodule output; submodule required.
4 `SET_HAND_GUIDING`	Core Output hand-guiding flag.
5 `MOVE_JOINT`	JointPose IO Submodule output; submodule required.
6 `MOVE_CARTESIAN`	CartesianPose IO Submodule output; submodule required.
7 `START_FLOW`	Core Output flow UUID; invalid UUID returns `REJECTED_VALUE`.
8 `PAUSE_FLOW`	No payload; implemented.
9 `RESUME_FLOW`	No payload; implemented.
10 `STOP`	No payload; implemented.
11 `SET_TOOL`	Core Output tool UUID; NIL clears the tool.
12 `CLAIM_CONTROL`	No payload; obtain PLC authority. See Single Point of Control.
13 `ACK_ERROR`	No payload; acknowledge the latest backend error. See Acknowledge an error.

Missing Payload, CartesianPose or JointPose IO returns `REJECTED` for the associated command. IDs not listed in the table are invalid. `Flow_action` and `Movement_action` do not exist; `cmd_id` selects the behavior directly. Core Output payload fields are defined in the Core Output offset table.

3.7.2 Table: last_req_status Values

Value	Name	PLC interpretation
0	NONE	no result
1	ACCEPTED	queued; not final
2	EXECUTING	running; not final
3	DONE	final success
4	REJECTED	final: unavailable, disallowed or not implemented
5	REJECTED_VALUE	final: invalid value, UUID, object or range
6	REJECTED_UNAUTHORIZED	final: PLC does not hold control authority
7	ABORTED	final: unexpected execution failure

EXECUTING is defined by the interface; the current backend may transition directly from ACCEPTED to a final result.

3.7.3 Acknowledge an error

Send ACK_ERROR through the normal command handshake while the PLC holds control. It acknowledges the latest backend error message; it does not reset the underlying robot or safety fault. The legacy `acknowledge_error` flag is not evaluated by this command. If no error message is available to acknowledge, the command returns REJECTED.

3.7.4 Legacy operation-mode byte

Core Output retains the UDT name `requested_operation_mode` for layout compatibility. The backend treats it as reserved and ignores it. Write zero. There is currently no PROFINET command for changing the SRCI operation mode.

3.8 AuxVariables

AuxVariables are normal cyclic process data and never use `cmd_id` or `cmd_seq`. Each plugged Aux module transfers one complete 64-byte bank.

3.8.1 Table: AuxVariable Directional Bank Mapping

Bank within one direction	Backend logical range
0	`0..63`
1	`64..127`
2	`128..191`
3	`192..255`

Input and Output banks are numbered independently. For each direction, sort the active modules by physical slot; the first is bank `0`, the second bank `1`, and so on. For example, an Aux Input in slot 2 and an Aux Output in slot 3 both map to logical bytes `0..63` in their respective directions. TIA `%I` and `%Q` addresses are independent of this mapping and need not be contiguous.

3.8.2 Table: AuxVariable Types

Type and size	PLC representation and wire behavior
Integer, 4 bytes	`DInt`; signed big-endian value
Float, 4 bytes	`Real`; IEEE-754 big-endian value
Boolean, 2 bytes	`Bool` plus `pad_byte`; explicit padding
JointPose, 32 bytes	`RobCo_JointPose`; eight `Real` values
CartesianPose, 24 bytes	`RobCo_CartesianPose`; XYZ and RZ/RY/RX
String36, 36 bytes	`Array[0..35] of Char`; Windows-1252, zero-padded

By design, every auxiliary string is a fixed-width String36 and can contain at most 36 characters. Do not use Siemens `String[36]`, which adds two management bytes. RobFlow truncates longer device-to-PLC values to their first 36 characters before encoding and zero-pads shorter values. PLC-to-device values therefore contain at most 36 characters. String36 can transfer a UUID in canonical textual form.

3.8.3 Typed overlay

The RobFlow export defines one directional UDT per configured bank. The checked-in placeholder demonstrates this 64-byte composition:

Relative byte	Input / Output field and type
0..3	`profIn` / `profOut`: `DInt`
4.0	`profIn2` / `profOut2`: `Bool`
4.1..4.7	unused bits: zero
5	`PadByte`: `Byte`
6..37	`JointAngles_Variable4` / `JointAngles_Output4`: `RobCo_JointPose`
38..61	`CartesianPose_Variable5` / `CartesianPose_Output5`: `RobCo_CartesianPose`
62..63	`Reserved`: two bytes

String36 is supported but does not fit this particular 62-byte composition.

Use these addressing rules:

1. Place one exported UDT at the actual TIA address of its Aux module.
2. For non-contiguous module addresses, use one physical tag per module and copy it to or from the corresponding directional bank.
3. Never overlay multiple banks as one contiguous area unless the TIA address assignment is actually contiguous and follows the direction-specific slot order.

3.9 Commissioning and Troubleshooting

3.9.1 Commissioning sequence

1. Verify the installed GSDML revision and station name.
2. Verify that online modules match Device view and have good IO status.
3. Confirm each PLC tag starts at the address assigned to its submodule.

4. Start with Core Data, obtain PLC authority using `CLAIM_CONTROL`, and check the matching result and `control_external_active` feedback.
5. Perform `SET_BRAKES` using a new `cmd_seq`; confirm the command result and brake feedback.
6. Add optional submodules before testing their commands.
7. Verify every Aux bank independently, including sparse physical slots.
8. Check reconnection and PLC STOP -> RUN before enabling automatic commands.

Useful Core Data watch values:

- `robco_core_out.flags.brakes_closed`
- `robco_core_out.command.cmd_id` and `robco_core_out.command.cmd_seq`
- `robco_core_in.status.ack_seq`, `last_req_seq` and `last_req_status`
- `robco_core_in.flags.brakes_closed`
- `robco_core_in.flags.control_external_active`
- `robco_core_in.flags.control_local_active`
- `robco_core_in.flags.enabling_device_pressed`
- `robco_core_in.flow_state`, `robot_state` and `safety_state`

3.9.2 Troubleshooting

Symptom	Check
Device does not connect	station name, DCP assignment, network and GSDML revision
Core values are shifted	old UDT or wrong Core Data start address
Optional command returns `REJECTED`	required Payload, CartesianPose or JointPose submodule
Command is not processed	`cmd_seq` unchanged or wrong response sequence read
Command returns `REJECTED_VALUE`	range, UUID, object or payload value
Command returns `REJECTED_UNAUTHORIZED`	claim PLC authority and confirm `control_external_active` before retrying
Aux value is in the wrong bank	direction-specific slot order and frontend bank offset
Typed Aux value is corrupt	direction, offset, UDT composition and padding

Symptom	Check
String36 is corrupt	use 36 `Char` bytes, not Siemens `String[36]`
No PLC control authority	`control_external_active` must be true

3.10 Value Tables

`flow_state`, `robot_state` and `safety_state` contain concrete values and abstract groupings:

- **Concrete** identifies one exact publishable state.
- **Abstract grouping** is a numeric parent for related states. A value ending in one zero covers its ten-value range; a value ending in two zeroes covers its hundred-value range. This is a range convention, not a bit mask.
- **Abstract/temporary** may also be published while a specific state is not yet available.

PLC logic must handle the exact numeric value received, including group values.

3.10.1 Table: flow_state Values

This field publishes the backend's `OperationMode` values. It reports the flow/teach mode and is independent of `cmd_id`. The backend forwards these values without a PROFINET-specific renumbering. The Core Output `requested_operation_mode` field does not select a state; see Legacy operation-mode byte.

Value	Name	Classification
100	`TEACH`	concrete
200	`FLOW`	abstract grouping
210	`FLOW_CONTINUOUS`	abstract grouping
211	`FLOW_CONTINUOUS_RUNNING`	concrete
212	`FLOW_CONTINUOUS_RESUME`	concrete
220	`FLOW_CONTINUOUS_PAUSED_STATES`	abstract grouping
221	`FLOW_CONTINUOUS_PAUSED_ON_TRAJECTORY`	concrete
222	`FLOW_CONTINUOUS_PAUSED_NOT_ON_TRAJECTORY`	concrete

Value	Name	Classification
500	ERROR	concrete
600	CRITICAL	concrete

3.10.2 Table: robot_state Values

Value	Name	Classification
100	IDLE_STATES	abstract grouping
101	DISABLED	concrete
102	SWITCHED_ON	concrete
103	IDLE	concrete
104	TRANSITION	concrete
110	IDLE_PAUSED_STATES	abstract grouping
111	IDLE_PAUSED_ON_TRAJECTORY	concrete
112	IDLE_PAUSED_NOT_ON_TRAJECTORY	concrete
200	MOVING	abstract/temporary
210	MOVING_COMMAND	abstract grouping
211	MOVING_PTP	concrete
212	MOVING_LINEAR	concrete
213	RESUME	concrete
220	DECELERATING	concrete
221	STREAMING	concrete
222	ROS_CONTROL	concrete
230	MANUAL_MOVEMENT	abstract grouping
231	JOGGING	concrete
232	HAND_GUIDING	concrete
500	ERROR	abstract grouping
510	CONNECTION_ERROR	abstract grouping

Value	Name	Classification
511	BACKEND_CONNECTION_ERROR	reserved/completeness
512	CONTROLLER_CONNECTION_ERROR	concrete
513	ROBOT_CONNECTION_ERROR	concrete
514	UNKNOWN_CONTROLLER_API_VERSION_ERROR	concrete
515	INVALID_CONTROLLER_API_VERSION_ERROR	concrete
520	MOVEMENT_ERROR	abstract grouping
521	CLOSE_TO_SINGULARITY_ERROR	concrete
522	LINEAR_MOTION_ERROR	concrete
523	VELOCITY_ERROR	concrete
524	INVALID_APPROACH_MODE_ERROR	concrete
525	ANGLE_OUT_OF_LIMIT_ERROR	concrete
530	MODULE_ERROR	grouping/concrete fallback
531	MODULE_OVERHEAT_ERROR	concrete
600	CRITICAL	concrete

3.10.3 Table: safety_state Values

Value	Name	Classification
100	MANUAL	concrete
200	AUTO	concrete
300	SAFE_STOP	abstract grouping
301	SAFE_STOP_0	concrete
302	SAFE_STOP_1	concrete
303	SAFE_STOP_2	concrete
304	INVALID	concrete; invalid safety-mode selection

Value	Name	Classification
500	ERROR	concrete
501	NOT_CONNECTED	concrete