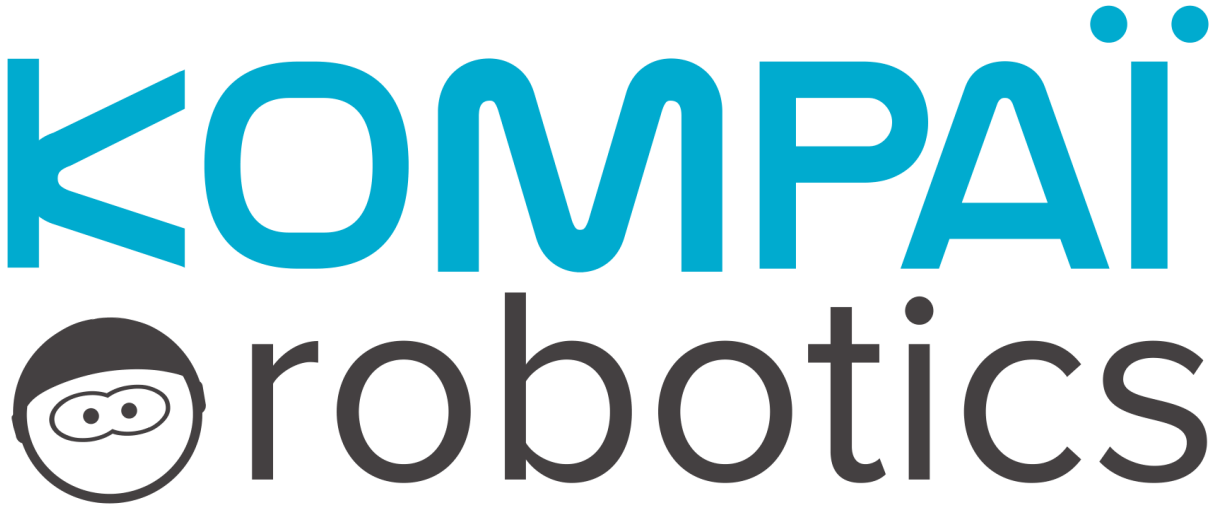


PURE Communication Manual

Release 5.0

Robosoft

Feb 18, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

Welcome to the PURE communication manual!

PURE is the name of Robosoft's low level control software. It is designed to perform tasks such as actuator control, sensor data acquisition, feedback control and robot supervision.

This manual describes the network protocol used to access PURE controllers functionalities. It is the standard way to interface with Robosoft robots low level functionalities.

1.1 Who should read it?

This manual is targeted at software developers who want to write their application by interfacing directly with Robosoft low level controller, PURE. There are several cases where this might be suitable:

- You need real-time behaviour and performance:

Typically, a PURE controller executes its control tasks every 10 milliseconds. It's possible to synchronize your application with PURE through this protocol, or even synchronize PURE with your application, by triggering yourself its control cycles!

- You have a system which is not supported by Robosoft:

Currently, our high level applications and the software we provide are developed using Microsoft Robotics Developer Studio.

1.2 Which prerequisites do I need?

This manual supposes that the developer has its own development environment, and is able to use a network API, like Winsock under Windows.

There are several implementations available, which may help you getting started:

- In RobuBOX, in the form of MRDS services. The code is open source and available under the LGPL (Lesser General Public License).
- As standalone C++ code, for Windows.
- As standalone C++ code, for Linux.
- As standalone C# code, for Windows.

1.3 What does it contain?

It contains:

- The communication protocol that allows to use PURE functionalities
- The description of standard interfaces available through the communication protocol
- Procedures to configure a PURE controller

It does not contain information specific to each robot, like the PURE services available, or the kind of hardware installed. These information can be found in the robot manual, available through its technical documentation page.

1.4 How to read this manual?

It is recommended to go through *chapter 2* a first time to get the basics of the protocol. The *One Axis Robot* example should give you a good idea of how to get started.

It is then possible to have a look at the code examples, or start writing your own implementation, while using *chapter 4* as a reference for application services. This chapter should be used with the robot manual, which contains specific details about available services and how they map to its hardware.

Chapter 5 is meant to be used when specific configuration tasks are required. It contains procedures to upload applications, update the system, and configure the IP address.

1.5 Changes from previous versions

1.5.1 release 5.0

Changes:

- Protocol: Request identifier
- Step: Added last command date
- Trajectory: Added last command and trajectory date

New:

- Added Anticollision service
- Added Signal service
- Added Statistics service

1.5.2 release 4.1

New:

- Added Gyroscope service

1.5.3 release 4.0

1.5.4 release 3.1

Changes:

- Trajectory: Added Replace request

New:

- Added Step control service

1.5.5 release 3.0

Changes:

- Directory: Suppressed number of entries in the GET request
- Diagnostic: Suppressed number of entries in the GET request
- Drive: Suppressed number of entries in the GET request
- Telemeter: Suppressed number of entries in the GET request
- Differential: Added Enable field in inbound notification
- Differential: Added Status field in outbound notification

New:

- Arm: Added service description

CHAPTER

TWO

COMMUNICATION PROTOCOL

2.1 Basic principles

The protocol was designed to be modular, in order to be reusable on the majority of our robots, but also exhaustive, to cover the highest possible number of use cases, in terms of control or data acquisition.

This led to the definition of two communication mechanisms:

- **Request/Response**: it is intended for punctual and/or asynchronous operations, such as retrieving robot properties, adding points in a trajectory buffer, etc.

  Requests are always issued by the high level controller. The PURE controller always sends back a Response for each request it receives.

- **Notification**: it is intended for synchronous operation, such as retrieving sensor data, or feeding a control loop.

  Notifications are one-way unconfirmed messages. They can be either sent or received by the PURE controller. When they are sent by the PURE controller, they are called **outbound** or **transmit** notifications. When they are received, they are called **inbound** or **receive** notifications.

These mechanisms are used to access the functionalities of PURE, which are organized in **services**:

- A **service** is similar to a type. It defines data structures and the way to access them using **Request** and **Notification** mechanisms.

  For example, basically all robots have some kind of motion actuators. They are exposed through the **Drive** service. This service defines a **Request** to read the drives properties (position limits, speed limits, ...). It also defines an **inbound notification** to set the drives targets (position, speed, ot torque), and an **outbound notification** to inform the user about the actual position, speed and torque.

- An **instance** of a service exposes an available function of the robot.

  If we have have a robot with two laser range finders, then the PURE controller will have two laser service instances.

There are two kind of services:

- System services, which expose system configuration.

  This includes the **Directory service**, which lists the services instances available on a PURE controller, and the **Notification service**, which allows to (de)activate outbound **Notifications**.

- Application services, which provide the actual robotic functionalities.

  A robot will most often have at least a **Drive service**, which exposes its axis, and for a mobile robot it will have a **Car service** or a **Differential service** which provides basic axis synchronization.

2.2 Practical considerations

2.2.1 Transport layer

The protocol described in this manual supposes the existence of a so called "Transport layer", which is responsible for transmitting data between the PURE controller and a client computer

Note: Currently, the only transport supported by PURE is the User Datagram Protocol (UDP). The UDP header is never mentionned, although it is used for the purposes listed below.

The PURE protocol assumes that the transport provides the following features:

- Addressing of a specific PURE controller. For UDP, it is through its IP address
- Determine the size of the frame. This is used for variable length messages, like those containing arrays or strings.
- Check the integrity of the messages.

Note: This protocol has been designed to be transport agnostic. Support for additional transport layers may be added if there are some specific needs. For example, it would be possible to implement a serial transport over a RS232 link.

2.2.2 Data types

The following table lists the base types used throughout this manual.

Note: All data types longer than one byte are encoded in little endian.

Note: Signed integer types are encoded in 2's complement.

Note: Floating point types are encoded according to IEEE 754 standard

Note: Character strings are treated as *Byte* arrays. They are **NOT** null terminated.

Table 1: Basic data types

Type	Description
Byte	An 8 bit field.
Int16	16 bit signed integer.
UInt16	16 bit unsigned integer.
Int32	32 bit unsigned integer.
UInt32	32 bit unsigned integer.
Float32	32 bit floating point number.
Float64	64 bit floating point number.

2.2.3 Units

All numeric values use SI units, unless stated otherwise.

The following table recapitulates them for quantities commonly used by PURE.

Table 2: Common units

Type	Unit	Symbol
Time	*seconds*	*s*
Distance	*meters*	*m*
Angle	*radians*	*r*
Linear speed	*meters per second*	*m/s*
Linear acceleration	*meters per square second*	m/s^2
Rotation speed	*radians per second*	*r/s*
Rotation acceleration	*radians per square second*	r/s^2
Torque	*Newtons meter*	*N.m*
Force	*Newtons*	*N*

2.3 Messages format

We describe here the format of request, response and notification messages.

2.3.1 Requests/responses

Message structure

The following tables show the structure of request and response messages:

Table 3: Request format

Byte	1	2	3 - 4	5 . . . N
Content	Identifier	Action	Target	Data

Table 4: Response format

Byte	1	2	3 - 4	5	5 . . . N
Content	Identifier	Action	Target	Result	Data

A response message is sent for each request message.

The response Identifier, Action and Target fields are copied from the request message.

Identifier field

This is a *Byte* which **MUST** be different from 0x00 and 0xFF (they are reserved for notifications).

This field is used primarily to indicate that the message is a request. This is indicated by the fact that the value of the field is different from 0x00 and different from 0xFF.

Its value will be used in the corresponding response message, allowing to check that it matches the request. Typically, a client implementation can increment this field at each request, and wrap around when reaching the value 0xFE.

It is also used to handle request or response transmission errors.

- If a client sends a request and doesn't receive a response, it can resend the request with the same identifier.
- When the server receives a request with a new identifier, it processes it and stores the response.
- If the server receives a request for a service instance with the same identifier, it immediately answers with the stored response.

This allows retrying state changing operations without unexpected results, like inserting twice a trajectory.

Action field

It specifies the operation to be performed on the service.

There is a defined set of actions, which carry a semantic meaning, to ease the understanding of their result. These actions are given in the following table, with examples.

Note: All actions are not supported by all services. Typically the GET request is defined for each service, but the use of the other actions depends on the needs of the services. The supported actions are documented in the service specific documentation section.

Table 5: Action codes

Action	Code	Usage example
GET	0x00	For the drive service, returns the physical properties of the axis (speed, position, etc.).
QUERY	0x01	For the I/O service, queries a friendly name for an input or output.
REPLACE	0x02	For a localization service which is based on odometry, replaces the current position.
UPDATE	0x03	For a localization service, adds an observation to be integrated in the solution.
INSERT	0x04	For a trajectory service, adds points in the following algorithm buffer.
DELETE	0x05	For a trajectory service, clears the following buffer.

Target field

It specifies which service instance should handle the request.

Data field

This field contains the optional service specific data. See the corresponding service documentation for more information.

Result field

This field is present only in response messages, and indicates if the request was processed correctly.

There are some codes that are common to all services. They are listed below.

Table 6: Result codes

Code	Name	Description
0x00	Success	The request has been successfully processed.
0x01	UnknownTarget	The Target field does not match any service instance.
0x02	ActionNotSupported	The service doesn't implement the specified action.
0x03	UnknownAction	The action field value is not one specified in the table above.
0x04	InvalidLength	The request message is smaller than the header size.
0x05	InvalidData	The target service has detected a problem in the data contained in the request.

Services may return other error codes. They are documented in the service section.

2.3.2 Notifications

Message structure

The following table gives the structure of inbound and outbound notifications.

Table 7: Inbound notifications

Byte	1	2 - 3	4 . . . N
Content	Identifier	Target	Data

Table 8: Outbound notifications

Byte	1	2 - 3	4 . . . 11	12 . . . N
Content	Identifier	Source	Timestamp	Data

Identifier field

This field always has the value 0xFF.

Source/Target field

This field indicates which service instance sent the notification (if it is outbound) or which should receive it (if it is inbound).

Timestamp field

This is the date of the notification content, in control cycles number since system start. The control cycle is by default 10 milliseconds.

Data field

This is the service specific data. This data is documented in the service specific sections, in *chapter 4*.

2.4 Example: A one axis robot

2.4.1 Description

Until now, we've seen a description of each piece of the protocol. We are now ready to put everything together and do something useful!

We will study the case of a theoretical robot, made of a single axis.

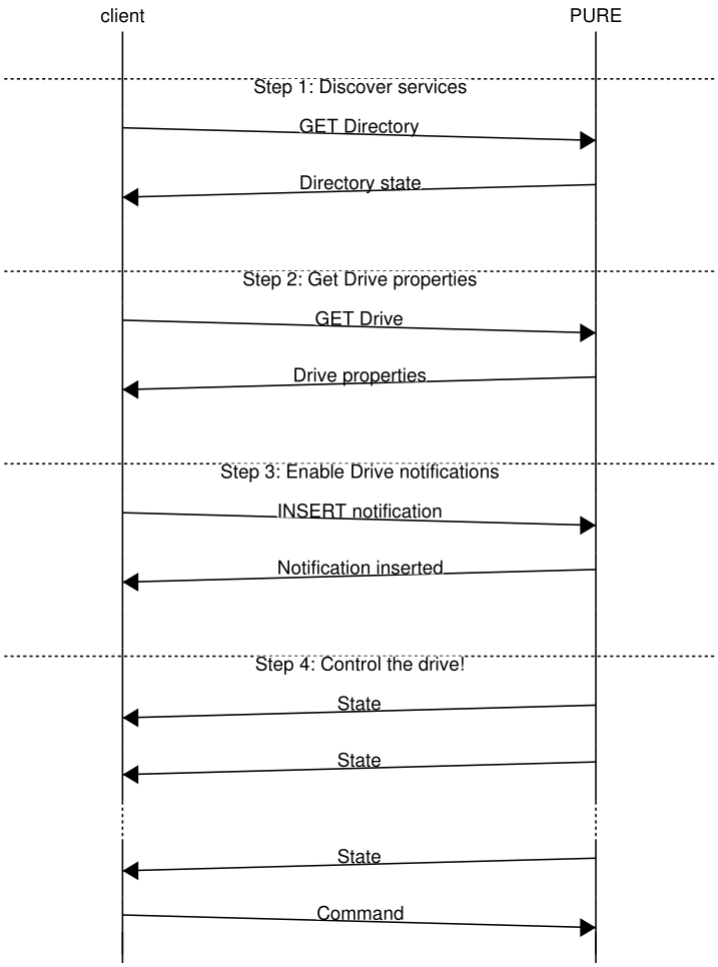
The PURE controller will be running the following services instances:

Table 9: Services instances

Instance	Service
0x00	Directory
0x01	Notification
0x02	Drive

We detail the steps needed to setup the communication in the following sub sections. These steps are summarized by the following messages sequence, exchanged between PURE and the client.

Note: Step 1 and 2 are not necessary if you already know the Drive service instance (which would be specified in the robot manual) Step 3 is necessary only if you need to know the state of the drive.



2.4.2 Step 1: Discovering the PURE controller services

Here we suppose that we have no idea about what the PURE controller is doing. We only have its IP address and the UDP port it is listening on.

To discover the controller capabilities, we're going to use the Directory service. This service always has an instance running with target number 0 on all PURE controllers. This gives us an entry point to start interacting with the controller.

We're going to issue a GET request to the directory service, which will give us all service instances.

Building the request

Lets list the elements we need to build our request message:

- Id: This is our first request, so we're going to use 1. 0x01
- Action: We want a GET , so check the actions code table 0x00
- Target: This is number 0, coded in 16 bits. 0x00 0x00
- Data: We have no data to send.

We're ready to send our request: call the UDP socket, and send the following 4 bytes buffer:

```
| Id | Action | Target
| 01 | 00     | 00 00
```

PURE will answer with this message:

```
| Id | Action | Target | Result | Data
| 01 | 00     | 00 00  | 00     | 00 00 00 00 | 01 00 01 00 | 09 40 02 00
```

Decoding the response

So, let's break down this into pieces! This gives us:

- Id: 0x01
- Action: 0x00
- Target: 0x00 0x00

These three fields of the response always match exactly those in the request. This allows us to make sure this is the response that matches our request.

- Result: 0x00

This means the request was successful.

- **Data**
  - 0x00 0x00 - 0x00 0x00 Directory service (0x0000)- Instance 0
  - 0x01 0x00 - 0x01 0x00 Notification service (0x0001) - Instance 1
  - 0x09 0x40 - 0x02 0x00 Drive service (0x4009) - Instance 2

We now know what we can do with the PURE controller!

We could now go to the next section.

There is however a litle bonus; we can get a friendly name for each service, through a QUERY request.

Getting a friendly name

We issue a QUERY to the Directory service with the Data field containing an instance number, for example 2 for the Drive service:

```
02 01 00 00 02 00
```

PURE answers:

```
02 01 00 00 00 44 72 69 76 65
```

44 72 69 76 65 is the ASCII code for Drive !

2.4.3 Step 2: Getting the drive properties

Now that we know the robot has a Drive service, we want more information about it. How many drives are there? What are their properties?(Maximum/minimum positions, speeds, acceleration)

We can retreive all these information through a GET request to the *Drive service*:

```
03 00 02 00
```

It's the same as the GET for the directory, except for the target instance field.

PURE answers (it's only one message, but split on several lines because of its length):

```
03 00 02 00 00  \
   01 01 00 00 80 3F 00 00 80 BF \
   00 00 00 40 00 00 00 C0 00 00 20 41 \
   00 00 00 00 00 00 00 00
```

Let's break the packet to understand it:

- 03 00 02 00 00 This the response header we are now familiar with.
- **The rest is the Drive service specific data, as described in its *section*.**
  - 01 : This is an angular drive, meaning all the numeric values will refer to angles.
  - 01 : The default mode of the drive is velocity mode.
  - 00 00 80 3F: The maximum position is 1 radian.
  - 00 00 80 BF: The minumum position is -1 radian.
  - 00 00 00 40: The maximum speed is 2 radians/second.
  - 00 00 00 C0: The minimum speed is -2 radians/second.
  - 00 00 20 41: The maximum acceleration is 10 r/s^2.
  - 00 00 00 00 00 00 00 00: The maximum and minimum torque are 0 N.m.

The fact that the limit torques are 0 means that the torque mode is not supported on this drive. On mobile robots for example, the position limits are usually 0 for traction motors, because there is no support for position mode.

2.4.4 Step 3: Enabling outbound notifications

Now that we know a little bit more on that drive, we probably want to know its state. This is were notifications enter in action. Outbound notifications are disabled by default. They can be enabled through the Notification service, using an INSERT request.

This request needs two arguments:

- The instance for which we want to enable notifications; here it's 0x0002.
- The period at which we want the notifications, in control cycles. Let's say we want a notification one cycle out of 5, this gives 5.

We can now build our request:

```
04 04 01 00 02 00 05
```

PURE will answer:

```
04 04 00 01 00
```

or:

```
04 04 00 01 11
```

if the notification is already active. The 0x11 is an error code specific to the Notification service.

From now on, if we read data on our socket, we will receive the messages with the following data:

```
FF 02 00 00 01 00 00 00 00 00 00 \
   01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FF 02 00 05 01 00 00 00 00 00 00 \
   01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FF 02 00 0A 01 00 00 00 00 00 00 \
   01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

FF 02 00 14 01 00 00 00 00 00 00 \
   01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
```

The field which changes is the timestamp. The rest of the message says that the drive is enabled, in velocity mode, its target speed is zero, and its current position, speed and torque are all zero.

We are now ready to control our drive.

2.4.5 Step 4: Controlling the drive

This is the most simple step; all we have to do is build a message with the desired control mode and target.

Let's say we want the drive to turn at 1 radian/second.

We need the following field in the inbound notification:

- Enable the drive: 01
- Velocity mode: 01
- Control target: 00 00 80 3F

This gives us this frame:

```
FF 02 00 01 01 00 00 80 3F
```

After sending this message, we should see the state notifications evolve like this:

```
FF 02 00 00 02 00 00 00 00 00 00 \
   01 01 00 00 80 3F CD CC 4C 3C 00 00 00 3F 00 00 00 00

FF 02 00 05 02 00 00 00 00 00 00 \
   01 01 00 00 80 3F CD CC 4C 3D 00 00 80 3F 00 00 00 00
...
```

The position and speed examples are calculated considering a perfect following taking the acceleration limit in account:

- Speed: 0.5, 1.0, ...
- Position: 0.0125, 0.05, ...

2.5 Services documentation

The next two chapters are a reference for *system* and *application* services (i.e. services that provide robotic functionalities).

Each of the sections in these chapter documents a service. Each section follows a similar pattern:

- The first subsection describes the purpose of the service.
- If the service supports some requests, each request has its own subsection.
- If the service supports notifications, there is a dedicated subsection for inbound and outbound notifications.

In the requests and notifications subsections, only the Data fields of the messages are documented. For requests subsections, there can be one or two Data field documentation; one for the request, or one for the response, or both (e.g. a GET request usually has only response data; the request data is empty).

The headers are not mentioned, except for the Result field for responses that have specific error codes.

The Data field is documented using a table that looks like the following one:

Offset	Type	Description
Start - End	TypeA	Purpose of the field.
Start - End	TypeB	...
...		

The first column gives the start byte and end byte offset of the field in the Data buffer.

The second column gives the type of the field. It can be either a base type listed *here*, or a service specific composite type. In the latter case, the definition of this type is given with the same format, just below the complete Data definition.

For example, The Drive service defines the data in response to a GET request as an array of **DriveProperties**.

The **DriveProperties** type is a composite type specific to the Drive service, and is defined just after the GET response data definition.

CHAPTER

THREE

SYSTEM SERVICES

3.1 Directory service

3.1.1 Overview

This service lists all the available services instances available on a PURE controller, including itself.

Table 1: Service info

Type	0x0000
Actions	GET, QUERY
Inbound	No
Outbound	No

There is always at least one instance running on a PURE controller, with instance number 0. This gives an entry point for discovery of the robot services.

The list of services instances and their types can be retreived with a GET request. For each instance in this list, it is possible to get a string description with a QUERY request.

3.1.2 GET request

This request returns a list of services instances.

Response data

Offset	Type	Description
0 - 3	DirectoryEntry	First service instance.
4 - 7	DirectoryEntry	Second service instance.
...		

DirectoryEntry type

Offset	Type	Description
0 - 1	UInt16	Service type.
2 - 3	UInt16	Service instance number.

The service type is the value given in the service info table, in the service documentation overview subsection. The service instance number is the value to use in the target field when sending or receiving messages.

3.1.3 QUERY request

This request returns a string description for an entry in the directory.

Request data

Offset	Type	Description
0 - 1	UInt16	Instance for which we want a description.

Response data

Offset	Type	Description
0 - ...	String	The description string.

3.2 Notification service

3.2.1 Overview

This service handles configuration of outbound notifications for all services.

Table 2: Service info

Type	0x0001
Actions	GET, INSERT, DELETE
Inbound	No
Outbound	No

An outbound notification configuration consists of two elements:

- The service instance which should send the notification.
- The notification mode; either periodic or event based.

3.2.2 GET request

This request returns the list of active outbound notifications.

Response data

Offset	Type	Description
0 - 2	NotificationEntry	First active notification.
3 - 5	NotificationEntry	Second active notification.
...		

NotificationEntry type

Offset	Type	Description
0 - 1	UInt16	Service Instance.
2 - 2	Byte	Notification mode; 0 = on change, 1 - 255 = period.

3.2.3 INSERT request

This request is used to activate a notification and configure its mode. It adds the configuration record to its list of active notifications, returned by a GET request.

Request data

Table 3: INSERT request data

Offset	Type	Description
0 - 2	NotificationEntry	Notification configuration. See description *here*

Error codes

This request might return the following errors:

Code	Description
0x10	Maximum number of notifications reached.
0x11	There is already an active notification matching the specified instance.

Note: Usually, error code 0x11 can be ignored. This means that the notification you want to activate is already active.

3.2.4 DELETE request

This request removes an active notification from the list, and deactivates it.

It must be used either to stop the notification emission, or prior to sending an INSERT request in order to change the notification mode (if the notification is already active).

Response data

Offset	Type	Description
0 - 1	UInt16	Service instance for which notifications should be disabled.

Error codes

Code	Description
0x04	The data does not contain two bytes (the size of an instance number)

3.3 Diagnostic service

3.3.1 Overview

This service gives information about the status of hardware devices controlled by PURE. It is used mainly for debugging and diagnostic purpose when something doens't work properly.

Table 4: Service info

Type	0x0002
Actions	GET, QUERY
Inbound	No
Outbound	Yes

The principle is to have one status word for each hardware device. Each status status word is a bitfield where each bit matches an error condition. When the word is zero, the hardware device is functionning properly. When not, the bit(s) set indicate the error condition.

3.3.2 GET request

This request retreives the list of all devices status words.

Response data

Offset	Type	Description
0 - 3	UInt32	Status of the first device.
4 - 7	UInt32	Status of the second device.
...		

3.3.3 QUERY request

This request returns a descriptive string for a device.

Request data

Offset	Type	Description
0 - 3	Int32	Index of the device for which we want a description.

Response data

Offset	Type	Description
0 - ...	String	The description string.

3.3.4 Outbound notification

The outbound notifiaction is used to inform the PURE client when the status of a device changes.

Notification data

Offset	Type	Description
0 - 3	UInt32	Status of the first device.
4 - 7	Float32	Status of the second device.
...		

3.4 Logger service

3.4.1 Overview

This service gives sends internal PURE log messages in outbound notifications.

Table 5: Service info

Type	0x0003
Actions	None
Inbound	No
Outbound	Yes

3.4.2 Outbound notification

Each notification contains one log message.

A log message has the following format:

```
Time [Source] Level : Message
```

The *Time* field is the PURE cycle at which the message was logged. The *Source* field is the component that logged the message. The *Level* field is one of INFO, ERROR or WARNING. *Message* is the actual log.

Notification data

Offset	Type	Description
0 - ...	String	The log message.

CHAPTER

FOUR

APPLICATION SERVICES

This chapter contains the application services reference.

This is where you can find out about the capabilities of a robot, the kind of data it provides and the type of commands it accepts.

4.1 Arm service

4.1.1 Overview

This service exposes a serial manipulator.

Table 1: Service info

Type	0xA002
Actions	GET, QUERY
Inbound	Yes
Outbound	Yes

This service offers the following functionalities:

- Inverse kinematic

The QUERY request is used to compute an inverse kinematic solution. Based on the 6 cartesian position coordinates contained in the request data, the PURE controller will try to compute some matching joint positions.

If a solution is found, the joints position are sent back in the response data. A solution might not always exist.

- Tool state

The outbound notification contains the cartesian position of the tool.

- Point to point motion

The inbound notification is used to specify target articular positions, along with desired speeds and accelerations.

Upon reception of an inbound notification, the PURE controller will compute trapezoidal motion profiles for all joints, such that the durations of the acceleration phase, constant speed phase, and deceleration phases are the same for all joints.

4.1.2 QUERY request

This request returns an inverse kinematic solution if it exists.

Request data

Offset	Type	Description
0 - 3	Float32	X coordinate, in meters.
4 - 7	Float32	Y coordinate, in meters.
8 - 11	Float32	Z coordinate, in meters.
12 - 15	Float32	Roll angle, in radians.
16 - 19	Float32	Pitch angle, in radians.
20 - 23	Float32	Yaw angle, in radians.

Response data

Offset	Type	Description
0 - 3	Float32	First joint position.
4 - 7	Float32	Second joint position.
...		

Error codes

Code	Description
0x11	There is no solution to the inverse kinematic problem.

4.1.3 Outbound notification

Notification data

Offset	Type	Description
0 - 3	Float32	X coordinate, in meters.
4 - 7	Float32	Y coordinate, in meters.
8 - 11	Float32	Z coordinate, in meters.
12 - 15	Float32	Roll angle, in radians.
16 - 19	Float32	Pitch angle, in radians.
20 - 23	Float32	Yaw angle, in radians.

4.1.4 Inbound notification

Notification data

Offset	Type	Description
0 - 11	AxisCommand	First axis command
12 - 15	AxisCommand	Second axis command
...		

AxisCommand Type

Offset	Type	Description
0 - 3	Float32	Target position
4 - 7	Float32	Maximum speed
8 - 11	Float32	Maximum acceleration

4.2 Battery Service

4.2.1 Overview

This service exposes the robot battery state. It is useful for autonomous operation, where you need to know when batteries need charging.

Table 2: Service info

Type	0x400D
Actions	GET
Inbound	No
Outbound	Yes

4.2.2 GET request

The GET request returns the battery properties.

Response data

Table 3: Response data description

Offset	Type	Description
0 - 3	Float32	Battery voltage, in Volts.
4 - 7	Float32	Battery capacity, Amperes.hour.
8 - 8	UInt8	Critical percentage.

The voltage and capacity fields are indicative. The critical percentage is used to determine the status of the battery, and is set in order to protect the hardware.

When the remaining power percentage falls under the critical level, the battery status changes to Critical.

4.2.3 Outbound notification

The outbound notification is used to send the remaining power.

Notification data

Offset	Type	Description
0 - 0	Byte	Status - 0 = Charging, 1 = Charged, 2 = Ok, 3 = Critical
1 - 1	UInt8	Remaining power percentage.

The status field gives the context of the battery:

- **Charging** means that the robot is plugged in a battery charger, and the charger is operating.
- **Charged** means that that the robot is still pugged in the charger, but it is not operating (i.e. the battery is charged at full capacity).
- **Ok** means that the battery is under normal operation.
- **Critical** means that the remaining power is below the Critical percentage threshold, and that the robot should be connected to a charger as soon as possible.

4.3 Car service

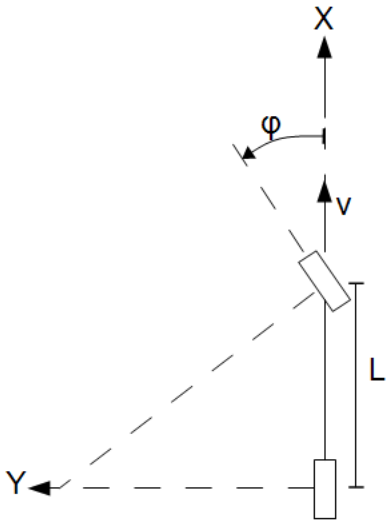
4.3.1 Overview

This service exposes a car like robot. A car robot is controlled through a speed and steering angle.

Table 4: Service info

Type	0x4003
Actions	GET
Inbound	Yes
Outbound	Yes

The picture below shows the model of a car robot, and how its frame (X, Y) is positioned.



This frame is the one referenced by localization data.

4.3.2 GET request

This request sends back the operation limits and geometry of the car robot.

Response data

Offset	Type	Description
0 - 3	Float32	Maximum speed, in *m/s*.
4 - 7	Float32	Minimum speed, in *m/s*.
8 - 11	Float32	Maximum steering angle, in *r*.
12 - 15	Float32	Minimum steering angle, in *r*.
16 - 19	Float32	Maximum acceleration, in m/s^2.
20 - 23	Float32	Maximum deceleration, in m/s^2.
24 - 27	Float32	Distance between the axles, in *m*.

4.3.3 Outbound notification

The outbound notification is used to retreive the actual speed and steering angle.

Notification data

Offset	Type	Description
0	Byte	Status - 0 = Disabled, 1 = Enabled, 2 = Error
1 - 4	Float32	Target speed, in *m/s*.
5 - 8	Float32	Current speed.
9 - 12	Float32	Target steering angle, in *r*.
13 - 16	Float32	Current steering angle.

4.3.4 Inbound notification

The inbound notification is used to send the target speed and steering angle.

Notification data

Offset	Type	Description
0 - 0	Byte	Enable - 0 = Disable, 1 = Enable
1 - 4	Float32	Target linear speed, in *m/s*.
5 - 8	Float32	Target steering angle, in *r*.

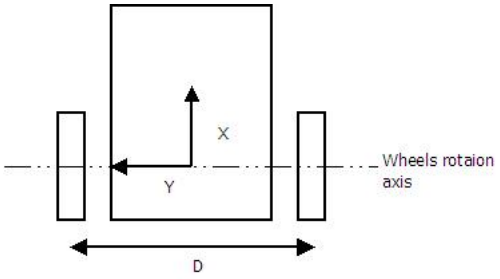
4.4 Differential Service

This service exposes a differential robot.

Table 5: Service info

Type	0x4005
Actions	GET
Inbound	Yes
Outbound	Yes

The figure below describes such a robot.



Differential robot

A differential robot is controlled through a linear speed and an angular speed. The linear speed component is obtained by applying the same speed to the left and right wheels. The angular speed component is obtained by applying speeds with opposite signs on the left and right wheels.

4.4.1 GET request

The GET request returns the properties of the differential robot.

These properties contain the operation limits and geometric decription of the robot.

Response data

Offset	Type	Description
0 - 3	Float32	Maximum linear speed, in *m/s*.
4 - 7	Float32	Minimum linear speed, in *m/s*.
8 - 11	Float32	Maximum angular speed, in *r/s*.
12 - 15	Float32	Minimum angular speed, in *r/s*.
16 - 19	Float32	Maximum linear acceleration, in m/s^2.
20 - 23	Float32	Minimum linear acceleration, in m/s^2.
24 - 27	Float32	Maximum angular acceleration, in r/s^2.
28 - 31	Float32	Minimum angular acceleration, in r/s^2.
32 - 35	Float32	Distance between the wheels, in *m*.

4.4.2 Outbound notification

The outbound notification is used to inform about the robot state.

Notification data

Offset	Type	Description
0 - 0	Byte	Status - 0 = Disabled, 1 = Enabled, 2 = Error
1 - 4	Float32	Target linear speed, in *m/s*.
5 - 8	Float32	current linear speed, in *m/s*.
9 - 12	Float32	Target angular speed, in *r/s*.
13 - 16	Float32	Current angular speed, in *r/s*.

4.4.3 Inbound notification

The inbound notification is used to set the target speeds for the differential robot control.

Notification data

Offset	Type	Description
0 - 0	Byte	Enable - 0 = Disable, 1 = Enable
1 - 4	Float32	Target linear speed, in *m/s*.
5 - 8	Float32	Target angular speed, in *r/s*.

4.5 Drive service

4.5.1 Overview

This service exposes an array of drives. It allows to get their states (status, position, speed, torque) through outbound notifications. The drives can be controlled (position, velocity or torque mode) through inbound notifications.

Table 6: Service info

Type	0x4009
Actions	GET
Inbound	Yes
Outbound	Yes

4.5.2 GET request

The GET request is used to retreive the drives properties. These properties define the drive operation limits. They are enforced by the PURE controller.

Response data

Offset	Type	Description
0 - 29	DriveProperties	Properties of the first drive
30 - 59	DriveProperties	Properties of the second drive
...	...	...

DriveProperties type

Offset	Type	Description
0 - 0	Byte	Type of the drive : 0 - Linear drive; 1 - Angular Drive
1 - 1	Byte	Default mode : 0 - Position; 1 - Velocity; 2 - Torque
2 - 5	Float32	Maximum position
6 - 9	Float32	Minimum position
10 - 13	Float32	Maximum speed
14 - 17	Float32	Minimum speed
18 - 21	Float32	Maximum acceleration
22 - 25	Float32	Maximum torque/force
26 - 29	Float32	Minimum torque/force

The units used for the drive limits depend on its type. For a linear drive, it will be *meters*, *m/s*, m/s^2, Newton, and for an angular drive it will be *radians*, *r/s*, r/s^2 and Newton.meter.

4.5.3 Outbound notification

The Drive service outbound notification is used to send its state to the client. It contains the actual operation status and values.

Notification data

Offset	Type	Description
0 - 17	DriveState	First drive state
18 - 35	DriveState	Second drive state
...	...	...

DriveState type

Offset	Type	Description
0 - 0	Byte	Actual mode; 0 - Position; 1 - Velocity; 2 - Torque
1 - 1	Byte	Status; 0 - Enabled; 1 - Disabled; 2 - Error
2 - 5	Float32	Actual control target.
6 - 9	Float32	Actual position
10 - 13	Float32	Actual speed
14 - 17	Float32	Actual force/torque

4.5.4 Inbound notification

The inbound notification is used to set the control mode and target.

Note: Depending on the hardware, a control mode change might not be immediate and therefore not reflected in the next outbound notification.

Notification data

Offset	Type	Description
0 - 5	DriveCommand	First drive command
6 - 11	DriveCommand	Second drive command
...	...	...

DriveCommand type

Offset	Type	Description
0 - 0	Byte	Enable; 0 - Disable; 1 - Enable
1 - 1	Byte	Control mode; 0 - Position; 1 - Velocity; 2 - Torque
2 - 5	Float32	Control target. Depending on the mode, it might be a position, speed or torque

4.6 Encoder Service

4.6.1 Overview

This service exposes an encoder device. An encoder measures a position, and optionally derives a speed from the position measurements.

Table 7: Service info

Type	0x400A
Actions	GET
Inbound	No
Outbound	Yes

4.6.2 GET request

The GET request returns the properties of the encoder.

Reponse data

Offset	Type	Description
0 - 0	Byte	Axis kind; 0 = linear axis, 1 = angular axis.
1 - 1	Byte	Encoder type; 0 = absolute, 1 = relative.

4.6.3 Outbound notification

The outbound notification is used to send the actual measurement of the encoder.

Notification data

Offset	Type	Description
0 - 3	Float32	Position
4 - 7	Float32	Speed

If the encoder axis is of linear kind, the position value will be in *meters*, or in *radians* otherwise. Similarly, the speed value will be in *m/s* or *r/s*.

4.7 GPS Service

4.7.1 Overview

This service exposes a GPS receiver.

Table 8: Service info

Type	0x400C
Actions	GET
Inbound	No
Outbound	Yes

The data available through this service matches the data found in a NMEA 0183 GPGGA message.

4.7.2 GET request

This request returns the position of the GPS device aantenna in the robot frame.

Response data

Offset	Type	Description
0 - 7	Float64	X coordinate of the GPS antenna, in *meters*.
8 - 15	Float64	Y coordinate of the GPS antenna, in *meters*.

4.7.3 Outbound notification

Notification data

Offset	Type	Description
0 - 7	Float64	Latitude, in *radians*.
8 - 15	Float64	Longitude, in *radians*.
16 - 23	Float64	Altitude, in *meters*.
24 - 24	GpsMode	Solution computation mode.
25 - 28	Int32	Number of satellites used in the computation.
29 - 36	UInt64	UTC time, in milliseconds.

GpsMode type

This type is an enumeration with the following values.

Note: See NMEA 0183 protocol description for more information on these values. The table below is only a quick reminder.

Value	Description
0	Invalid
1	SPS
2	Differential SPS
3	PPS
4	Fixed RTK
5	Floating RTK
6	Estimated
7	Manual input
8	Simulator
9	WAAS

4.8 Gyroscope Service

4.8.1 Overview

This service exposes a Gyroscope device. A gyroscope measures a rotation speed.

Table 9: Service info

Type	0x400E
Actions	
Inbound	No
Outbound	Yes

4.8.2 Outbound notification

Notification data

Offset	Type	Description
0 - 3	Float32	Rotation speed, in r/s^2.

4.9 I/O Service

4.9.1 Overview

This service gives access to digital and analog inputs and outputs.

Table 10: Service info

Type	0x4001
Actions	GET, QUERY
Inbound	Yes
Outbound	Yes

The values exposed by this service might as well corresponds to physical I/O or internal values of the PURE controller. The meaning of the values is detailed in the robot documentation.

Note: In the rest of this section, *AI* means *Analog input*, *AO* means *Analog output*, *DI* means *Digital input*, and *DO* means *Digital output*.

4.9.2 GET request

The GET request returns the numbers of I/O of different kinds (input, output, analog, digital).

Response data

Offset	Type	Description
0 - 3	Int32	Number of analog inputs.
4 - 7	Int32	Number of analog outputs.
8 - 11	Int32	Number of digital inputs.
12 - 15	Int32	Number of digital outputs.

4.9.3 QUERY request

The QUERY request returns a user friendly description of the specified I/O.

Request data

Offset	Type	Description
0 - 0	Byte	Type of the I/O (0 = AI, 1 = AO, 2 = DI, 3 = DO)
1 - 4	Int32	Index of the I/O

Response data

Offset	Type	Description
0 - ...	String	The I/O friendly name.

4.9.4 Outbound notification

This notification returns the values of the analog and digital inputs.

Notification data

Offset	Type	Description
0 - 3	Float32	First analog input
4 - 7	Float32	Second analog input
...	...	...
$4xN_{ai}$ -	Byte	First eight digital inputs
$4xN_{ai}$ + 1 -	Byte	Next eight digital inputs.
...	...	...

N_{ai} is the number of analog inputs.

4.9.5 Inbound notification

This notification sets the values of the analog and digital outputs.

Notification data

Offset	Type	Description
0 - 3	Float32	First analog output
4 - 7	Float32	Second analog output
...	...	...
$4xN_{ao}$ -	Byte	First eight digital outputs
$4xN_{ao}$ + 1 -	Byte	Next eight digital outputs.
...	...	...

N_{ao} is the number of analog outputs.

4.10 Laser service

4.10.1 Overview

This service provides measurements from a laser range finder.

Table 11: Service info

Type	0x4004
Actions	GET
Inbound	No
Outbound	Yes

A laser range finder is a sensor that measures an array distances at different angles, called a scan.

A pair (angle, distance) is called an echo.

4.10.2 GET request

This request returns the properties of the laser sensor.

Response data

Offset	Type	Description
0 - 3	Float32	X coordinate of the sensor, in the robot frame.
4 - 7	Float32	Y coordinate of the sensor, in the robot frame.
8 - 11	Float32	Orientation of the sensor, in the robot frame.
12 - 15	Int32	Number of echoes.

4.10.3 Outbound notification

Notification data

Offset	Type	Description
0 - 7	Echo	First measurement.
8 - 15	Echo	Second measurement.
...		

Echo type

Offset	Type	Description
0 - 3	Float32	Angle of the measurement, in *radians*.
4 - 7	Float32	Distance measured, in *meters*.

4.11 Localization Service

4.11.1 Overview

This service provides 2D localization data.

Table 12: Service info

Type	0x8002
Actions	GET, REPLACE, UPDATE
Inbound	No
Outbound	Yes

The localization provided usually refers to the origin of the robot frame. This is the middle of the wheels axis for a differential robot, or the middle of rear axle for a car robot.

4.11.2 GET request

This request returns the actual localization.

Response data

Offset	Type	Description
0 - 7	Float64	X coordinate.
8 - 15	Float64	Y coordinate.
16 - 23	Float64	Orientation.
24 - 24	UInt32	Status.

Status field

This field is a bit field which gives information about the localization data. If all bits are zeroed, the data is invalid.

Bit	Description
0	The solution has a metric accuracy.
1	The solution has a decimetric accuracy.
2	The solution has a centimetric accuracy.
3	The solution is computed using proprioceptive input.
4	The solution is computed using exteroceptive input.
5	An error was detected in the computation.

4.11.3 REPLACE request

This request is used to reset the localization. This is usually supported when the localization is only using proprioceptive input, like odometry.

See the robot manual to check if it is supported.

Request data

Offset	Type	Description
0 - 7	UInt64	Timestamp of the data
8 - 15	Float64	X coordinate.
16 - 23	Float64	Y coordinate.
24 - 31	Float64	Orientation.

4.11.4 Update request

This request is used to provide additional localization information not available to the PURE controller.

Unlike the REPLACE request, it's not a hard reset of the localization data.

For example, if the PURE controller only have odometry, it will accept data from this request and merge it with the dead reckoning estimation, for example using a Kalman filter.

Request data

Offset	Type	Description
0 - 7	UInt64	Timestamp of the data
8 - 15	Float64	X coordinate.
16 - 23	Float64	Y coordinate.
24 - 31	Float64	Orientation.

4.11.5 Oubound notification

This notification sends the actual localization data.

The data is the same as the GET respone data.

4.12 Step service

4.12.1 Overview

This service executes step motion commands. A step motion is either a rotation or translation (but not both at the same time) relative to the current position.

Table 13: Service info

Type	0x8003
Actions	UPDATE
Inbound	No
Outbound	Yes

4.12.2 UPDATE request

The UPDATE request allows setting a translation or rotation command.

The server will return in the response data the timestamp at which the request was received.

The last received command timestamp is specified in the outbound notification. This allows checking when the command is applied.

Request data

Offset	Type	Description
0 - 0	Motion	Motion type
1 - 4	Float32	Target distance
5 - 8	Float32	Maximum speed

The target distance is specified in meters for a translation, and in radians for a rotation. The maximum speed is specified in meters/second for translation or radians/second for a rotation.

The maximum speed specifies the maximum absolute value of the translation (or rotation) speed during the motion. It is guaranteed that it will not be exceeded, but not that it will be reached.

Motion type

Value	Description
0	Translation
1	Rotation
2	Stop

Response data

Offset	Type	Description
0 - 7	UInt64	Timestamp

4.12.3 Outbound notification

The Step service outbound notification is used to send its state to the client.

Notification data

Offset	Type	Description
0 - 7	UInt64	Last command timestamp
8 - 8	Motion	Requested motion type
9 - 12	Float32	Target distance
13 - 16	Float32	Maximum speed
17 - 20	Float32	Actual distance
21 - 21	Byte	Motion status (0 : In progress, 1 : completed)

4.13 Telemeter service

This service exposes an array of telemeters.

Table 14: Service info

Type	0x4008
Actions	GET
Inbound	No
Outbound	Yes

A telemeter is a sensor that measures a distance. Typically, ultrasonic sensors, or infrared sensors are available through this service.

4.13.1 GET request

This request returns an array of telemeters properties.

4.13.2 Response data

Offset	Type	Description
0 - 23	TelemeterProperties	First telemeter properties.
24 - 47	TelemeterProperties	Second telemeter properties.
...		

4.13.3 TelemeterProperties type

Offset	Type	Description
0 - 3	Float32	X coordinate of the telemeter in the robot frame.
4 - 7	Float32	Y coordinate of the telemeter in the robot frame.
8 - 11	Float32	Orientation of the telemeter in the robot frame.
12 - 15	Float32	Field of view of the telemeter.
16 - 19	Float32	Minimum measurable distance.
20 - 23	Float32	Maximum measurable distance.

Outbound notification

This notification contains the telemeters actual measurements.

4.13.4 Notification data

Offset	Type	Description
0 - 3	Float32	First telemeter distance measurement.
4 - 7	Float32	Second telemeter distance measurement.
...		

4.14 Trajectory service

This service provides an access to the input of a trajectory following algorithm.

Table 15: Service info

Type	0x8001
Actions	GET, INSERT, UPDATE, REPLACE
Inbound	No
Outbound	Yes

4.14.1 Description

Trajectory buffer

The principle of this service is to manage a trajectory buffer. Each element of this buffer is a segment associated with a target speed.

The trajectory following algorithm that uses the trajectory buffer is supposed to dequeue the segments as the robot moves.

The client can feed this trajectory buffer using the INSERT request, or the REPLACE request.

The trajectory following algorithm is also expected to manage the speed with respect to the target specified on each segement.

It is not guaranteed that the requested speed will be reached, but it is guaranteed that the actual speed will always be lower than the requested.

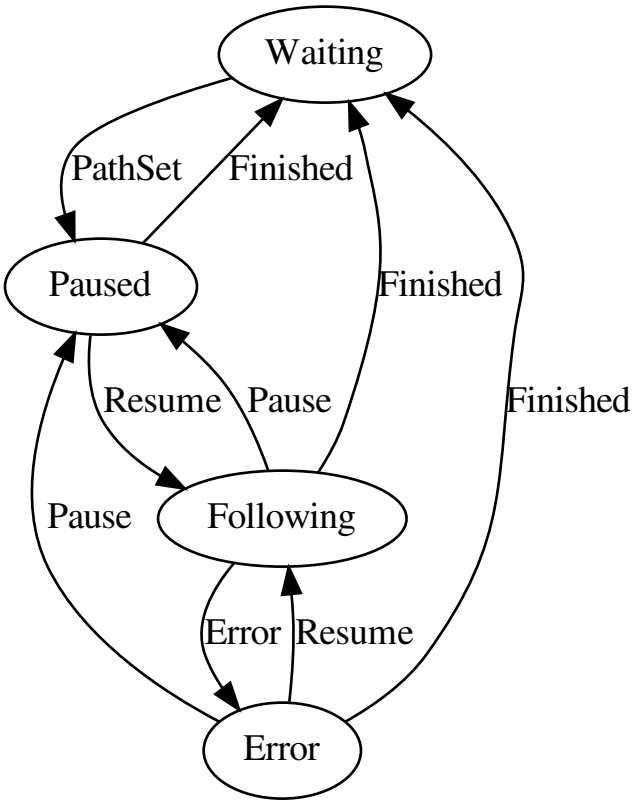
This is useful when specifying a lower speed for cornering. The client doesn't have to account for the deceleration that must occur before. It can just specify at a given point the maximum speed allowable, and the trajectory following will reduce the speed to respect the constraint.

Motion state machine

The motion is then defined by a state machine that takes commands either from the following algorithm or requests sent by the user.

This state machine is described in the diagram below.

Each ellipse is the state identified by its text, state transitions are represented by the directed arrows, and events that trigger the transitions as labels on the directed arrows.



- Waiting state :

There is no segment in the trajectory buffer.

This is the initial state.

You have to provide segments to get out of this state. This will raise the PathSet event.

You must use the INSERT request to provide segments.

- Paused state :

There is at least one segment in the trajectory buffer, and the lateral and orientation errors are within the allowed bounds. However, the robot has a target speed set to zero.

If the state is “Waiting”, and you insert some segments, the state will automatically switch to paused.

- Following state :

The same as the Paused state, except the algorithms is trying to reach the speed specified for the current segment.

It is possible to switch from “Paused” to “Following” using the UPDATE request, with the Resume command.

- Error state :

We decelerate until stopped with the emergency deceleration.

We will reach this state from the “Paused” or “Following” states if the lateral or orientation errors are too big.

The client can also force the transition to this state using the Halt command.

The transitions back to “Paused” or “Following” states are guarded until the error disappear.

The transition to the Waiting state is possible through a ResetPath command. This transition is not guarded.

4.14.2 GET request

This request returns the current state of the trajectory following controller.

Response data

Table 16: GET Response

Offset	Type	Description
0 - 7	UInt64	Timestamp
8 - 8	Byte	Status of the following
9 - 16	Float64	Distance covered
17 - 24	Float64	Lateral error
25 - 32	Float64	Orientation error

Table 17: Status values

Value	Description
0	Following : the robot is following a trajectory
1	Paused : the robot is following, but with a zero target speed
2	Waiting : there is no more trajectory
3	Error : An error was detected during the following

4.14.3 INSERT request

This request adds segments to the trajectory buffer.

Request data

Offset	Type	Description
0 - 39	Segment	First segment to insert.
40 - 79	Segment	Second segment to insert.
...		

Segment type

Offset	Type	Description
0 - 7	Float64	X coordinate of the segment start point.
8 - 15	Float64	Y coordinate of the segment start point.
16 - 23	Float64	X coordinate of the segment end point.
24 - 31	Float64	Y coordinate of the segment end point.
32 - 39	Float64	Target speed on the segment.

Error codes

Code	Description
0x11	The buffer is full. No more segments can be added.

4.14.4 Update request

This request is used to set commands for the trajectory control algorithm.

Request data

Offset	Type	Description
0 - 0	Command	Command that should be executed by the trajectory controller.

Command values

Value	Description
0 - Halt	Stops the robot with a strong deceleration.
1 - Resume	Starts the trajectory following.
2 - Pause	Stops the trajectory following with a comfort acceleration.
3 - Reset	Clears the trajectory following (resets the trajectory buffer and the distance)
4 - ResetPath	Resets the trajectory buffer.
5 - ResetDistance	Resets the distance covered.

4.14.5 REPLACE request

This request will replace the elements in the trajectory buffer with the ones provided in the request data.

Request data

Offset	Type	Description
0 - 39	Segment	First segment to insert.
40 - 79	Segment	Second segment to insert.
...		

4.14.6 Outbound notification

This is the same as the GET response, except for the timestamp that is already in the notification header.

Notification data

Table 18: Notification data

Offset	Type	Description
0 - 0	Byte	Status of the following
1 - 8	Float64	Distance covered
9 - 16	Float64	Lateral error
17 - 24	Float64	Orientation error

CHAPTER

FIVE

ADMINISTRATION PROCEDURES

5.1 Default configuration

PURE low level controllers have the following default configuration:

- IP Address: 192.168.1.2
  - Mask: 255.255.255.0
  - Gateway: 192.168.1.1
- UDP server on port 60000

5.2 Setting the IP address

The IP settings of the PURE controller can be changed through a command line utility program present on the controller itself. The instructions given here assume that it is done from a Windows computer.

- The first step is open a telnet session.

For this, you need to open a command prompt. ("Start Menu -> Run", then enter "cmd" and click "Ok").

At the prompt, type the following command:

```
telnet 192.168.1.2
```

You should get a welcoming screen from the controller.

- Use the netcfg.exe utility

In the telnet session, type the following command to change the address to 10.0.0.10/8, with a gateway address 10.0.0.1:

```
netcfg 10.0.0.10 255.0.0.0 10.0.0.1
```

After a few seconds, you will have a confirmation message.

- You can now reboot the controller for the changes to take effect.

This can be done either by powering off and on the controller or by using the following command, still in the telnet session:

```
devctl reboot
```

5.3 Updating the software

Robosoft might provide you some software updates from time to time.

There are two kind of updates:

- System update: this concerns the Windows CE operating system runtime which is used by PURE. This comes in the form of a single binary file, called NK.BIN.
- Application update: this concerns the PURE application itself. It comes in the form of a dll (exemple robulab.dll), and one or more file with the .pure extension.

5.3.1 Updating the application

For this operation, you will need a FTP client, such as Filezilla, or even the Windows Explorer built-in client.

- Connect to the device FTP server (anonymous login is allowed), and navigate to the directory \flashdisk\pure\

If you're using Windows Explorer, type ftp://192.168.1.2/flashdisk/pure in the address bar.

- You will find a file called autostart. Rename it to something else, like notautostart.
- Reboot the device.
- Reconnect to the FTP server.
- Delete the content of the directory \flashdisk\pure\ (except the notautostart file), and upload the update in place.
- Rename the notautostart file back to autostart.

NOTE: If you need to keep the previous version, you can also store it on the device in another directory, on the flashdisk. For exemple, flashdiskpurebackup.

5.3.2 Updating the system

- When you receive the Windows CE system update, extract the NK.BIN file from the ZIP archive somewhere on your computer.

For the example, we will suppose that the path is c:\path\to\image.

- For this operation you will need to open two command prompts. With the first one, open a telnet session (see first step of Changing the IP Settings).

When the telnet session is opened, type the following command:

```
devctl update
```

The prompt will block on this program. The device is now waiting for the new system image.

- In the second command prompt, type the following command:

```
tftp -i 192.168.1.2 put c:\path\to\image\nk.bin nk.bin
```

The file transfer should take a few seconds.

Warning: Once it's completed, wait for the device to reboot. Once the file download is complete, the file still needs to be written in the flash memory. If the system is stopped before it's completed, it may not be working anymore, and would need to be returned to us.