UNIVERSAL ROBOTS Support

REAL-TIME DATA EXCHANGE (RTDE) GUIDE

This is a guide on how to use the data synchronization protocol of the UR controller

Last modified on Dec 21, 2022

Created Date: October 10th, 2019.

This is a guide on how to use the data synchronization protocol of the UR controller e.g. when developing URCaps/UR+ for Universal Robots.

NOTE: All files are avalable for download at the bottom of this page.

Examples are valid for:
CB3 Software version: from 3.4 forward
e-Series Software versions: All versions
Note that newer software versions may behave differently.

The RTDE is available on CB3/CB3.1 from software 3.4, but specific features may only be available in newer software versions.

The RTDE synchronize external executables with the UR controller, for instance URCaps, without breaking any real-time properties. This document first describes the protocol and then provides a Python client reference implementation.

To have an overview of used ports on local host please read this post in the UR FORUM: https://forum.universal-robots.com/t/overview-of-used-ports-on-local-host/8889

REAL-TIME DATA EXCHANGE (RTDE)

INTRODUCTION

The Real-Time Data Exchange (RTDE) interface provides a way to synchronize external applications with the UR controller over a standard TCP/IP connection, without breaking any real-time properties of the UR controller. This functionality is among others useful for interacting with fieldbus drivers (e.g. Ethernet/IP), manipulating robot I/O and plotting robot status (e.g. robot trajectories). The RTDE interface is by default available when the UR controller is running.

The synchronization is configurable and can for example involve the following data:

- Output: robot-, joint-, tool- and safety status, analog and digital I/O's and general purpose output registers
- Input: digital and analog outputs and general purpose input registers

The RTDE functionality is split in two stages: a setup procedure and a synchronization loop.

On connection to the RTDE interface, the client is responsible for setting up the variables to be synchronized. Any combination of input and output registers that the client needs to write and read, respectively, can be specified. To achieve this the client sends a setup list of named input and output fields that should be contained in the actual data synchronization packages. The definition of a synchronization data package format is referred to as a recipe. There is a maximum limit of 2048 bytes to represent the list of inputs/outputs field names a client would like to subscribe to. In return the RTDE interface replies with a list of the variable types or specifies that a specific variable has not been found. Each input recipe that has been successfully configured will get a unique recipe id. The list of supported field names and their associated data types can be found below. When the setup is complete the data synchronization can be started and paused.

When the synchronization loop is started, the RTDE interface sends the client the requested data in the same order it was requested by the client. Furthermore the client is expected to send updated inputs to the RTDE interface on a change of value. The data synchronization uses serialized binary data.

All packages share the same general structure with a header and a payload if applicable. The packages used for the setup procedure generate a return message. The synchronization loop packages do not. Both client and server may at any time send a text message, whereby the warning level specifies the severity of the problem. The RTDE is available on port number 30004.

To get started we recommend using or modify the provided client sample written in Python. You can clone the repository at this link using git:

```
git clone https://github.com/UniversalRobots/RTDE_Python_Client_Library.git
```

Or in alternative you can download the release as a .zip package here.

KEY FEATURES

- Real-time synchronization: The RTDE generally generates output messages on 125 Hz. However, the real-time loop in the controller has a higher priority than the RTDE interface. Therefore, if the controller lacks computational resources it will skip a number of output packages. The skipped packages will not be sent later, the controller will always send the most recent data. Input packages will always be processed in the control cycle they are received, so the load for the controller will vary depending on the number of received packages.
- Input messages: The updating of variables in the controller can be divided into multiple messages. One can have one message to update everything or a message per variable or any division in between. There is no need for a constant update rate; inputs keep their last received value. Note: Only one RTDE client is allowed to control a specific variable at any time.
- Runtime environment: An RTDE client may run on the UR Control Box PC or on any external PC. The advantage of running the RTDE client on the Control Box is no network latency. However, the RTDE client and UR controller will compete for computational resources. Please make sure that the RTDE client runs with standard operating system priority. Computationally intensive processes, e.g. image processing, should be run on an external PC.
- Protocol changes: The RTDE protocol might be updated at any time by UR. To guarantee maximum compatibility for your RTDE client, RTDE clients can request the RTDE interface to speak a specific protocol version. Protocol additions / changes are explicitly denoted, otherwise version 1 is assumed.

FIELD NAMES AND ASSOCIATED TYPES

ROBOT CONTROLLER INPUTS

Name	Type	Comment	Introduced in version

speed_slider_mask	UINT32	0 = don't change speed slider with this input 1 = use speed_slider_fraction to set speed slider value	
speed_slider_fraction	DOUBLE	new speed slider value	
standard_digital_output_mask	UINT8	Standard digital output bit mask	
configurable_digital_output_mask	UINT8	Configurable digital output bit mask	
standard_digital_output	UINT8	Standard digital outputs	
configurable_digital_output	UINT8	Configurable digital outputs	
standard_analog_output_mask	UINT8	Standard analog output mask Bits 0-1: standard_analog_output_0 | standard_analog_output_1	
standard_analog_output_type	UINT8	Output domain {0=current[mA], 1=voltage[V]} Bits 0-1: standard_analog_output_0 | standard_analog_output_1	
standard_analog_output_0	DOUBLE	Standard analog output 0 (ratio) [0..1]	
standard_analog_output_1	DOUBLE	Standard analog output 1 (ratio) [0..1]	
input_bit_registers0_to_31	UINT32	General purpose bits This range of the boolean input registers is reserved for FieldBus/PLC interface usage.	
input_bit_registers32_to_63	UINT32	General purpose bits This range of the boolean input registers is reserved for FieldBus/PLC interface usage.	
input_bit_register_*X*	BOOL	64 general purpose bits X: [64..127] - The upper range of the boolean input registers can be used by external RTDE clients (i.e URCAPS).	3.9.0 / 5.3.0
input_int_register_*X*	INT32	48 general purpose integer registers X: [0..23] - The lower range of the integer input registers is reserved for FieldBus/PLC interface usage. X: [24..47] - The upper range of the integer input registers can be used by external RTDE clients (i.e URCAPS).	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0
input_double_register_*X*	DOUBLE	48 general purpose double registers X: [0..23] - The lower range of the double input registers is reserved for FieldBus/PLC interface usage.	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0

		X: [24..47] - The upper range of the double input registers can be used by external RTDE clients (i.e URCAPS).	
external_force_torque	VECTOR6D	Input external wrench when using ft_rtde_input_enable builtin.	3.3

ROBOT CONTROLLER OUTPUTS

Name	Type	Comment	Introduced in version
timestamp	DOUBLE	Time elapsed since the controller was started [s]	
target_q	VECTOR6D	Target joint positions	
target_qd	VECTOR6D	Target joint velocities	
target_qdd	VECTOR6D	Target joint accelerations	
target_current	VECTOR6D	Target joint currents	
target_moment	VECTOR6D	Target joint moments (torques)	
actual_q	VECTOR6D	Actual joint positions	
actual_qd	VECTOR6D	Actual joint velocities	
actual_current	VECTOR6D	Actual joint currents	
joint_control_output	VECTOR6D	Joint control currents	
actual_TCP_pose	VECTOR6D	Actual Cartesian coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz is a rotation vector representation of the tool orientation	
actual_TCP_speed	VECTOR6D	Actual speed of the tool given in Cartesian coordinates. The speed is given in [m/s] and the rotational part of the TCP speed (rx, ry, rz) is the angular velocity given in [rad/s]	
actual_TCP_force	VECTOR6D	Generalized forces in the TCP. It compensates the measurement for forces and torques generated by the payload	
target_TCP_pose	VECTOR6D	Target Cartesian coordinates of the tool: (x,y,z,rx,ry,rz), where rx, ry and rz is a rotation vector representation of the tool orientation	
target_TCP_speed	VECTOR6D	Target speed of the tool given in Cartesian coordinates. The speed is given in [m/s] and the rotational part of the TCP speed (rx, ry, rz) is the angular velocity given in [rad/s]	
actual_digital_input_bits	UINT64	Current state of the digital inputs. 0-7: Standard, 8-15: Configurable, 16-17: Tool	
joint_temperatures	VECTOR6D	Temperature of each joint in degrees Celsius	
actual_execution_time	DOUBLE	Controller real-time thread execution time	
robot_mode	INT32	Robot mode	

joint_mode	VECTOR6INT32	Joint control modes	
safety_mode	INT32	Safety mode	
safety_status	INT32	Safety status	3.10.0 / 5.4.0
actual_tool_accelerometer	VECTOR3D	Tool x, y and z accelerometer values	
speed_scaling	DOUBLE	Speed scaling of the trajectory limiter	
target_speed_fraction	DOUBLE	Target speed fraction	
actual_momentum	DOUBLE	Norm of Cartesian linear momentum	
actual_main_voltage	DOUBLE	Safety Control Board: Main voltage	
actual_robot_voltage	DOUBLE	Safety Control Board: Robot voltage (48V)	
actual_robot_current	DOUBLE	Safety Control Board: Robot current	
actual_joint_voltage	VECTOR6D	Actual joint voltages	
actual_digital_output_bits	UINT64	Current state of the digital outputs. 0-7: Standard, 8-15: Configurable, 16-17: Tool	
runtime_state	UINT32	Program state	
elbow_position	VECTOR3D	Position of robot elbow in Cartesian Base Coordinates	3.5.0 / 5.0.0
elbow_velocity	VECTOR3D	Velocity of robot elbow in Cartesian Base Coordinates	3.5.0 / 5.0.0
robot_status_bits	UINT32	Bits 0-3: Is power on | Is program running | Is teach button pressed | Is power button pressed	
safety_status_bits	UINT32	Bits 0-10: Is normal mode | Is reduced mode | Is protective stopped | Is recovery mode | Is safeguard stopped | Is system emergency stopped | Is robot emergency stopped | Is emergency stopped | Is violation | Is fault | Is stopped due to safety	
analog_io_types	UINT32	Bits 0-3: analog input 0 | analog input 1 | analog output 0 | analog output 1, {0=current[mA], 1=voltage[V]}	
standard_analog_input0	DOUBLE	Standard analog input 0 [mA or V]	
standard_analog_input1	DOUBLE	Standard analog input 1 [mA or V]	
standard_analog_output0	DOUBLE	Standard analog output 0 [mA or V]	
standard_analog_output1	DOUBLE	Standard analog output 1 [mA or V]	
io_current	DOUBLE	I/O current [mA]	
euromap67_input_bits	UINT32	Euromap67 input bits	
euromap67_output_bits	UINT32	Euromap67 output bits	
euromap67_24V_voltage	DOUBLE	Euromap 24V voltage [V]	
euromap67_24V_current	DOUBLE	Euromap 24V current [mA]	

tool_mode	UINT32	Tool mode	
tool_analog_input_types	UINT32	Output domain {0=current[mA], 1=voltage[V]} Bits 0-1: tool_analog_input_0 | tool_analog_input_1	
tool_analog_input0	DOUBLE	Tool analog input 0 [mA or V]	
tool_analog_input1	DOUBLE	Tool analog input 1 [mA or V]	
tool_output_voltage	INT32	Tool output voltage [V]	
tool_output_current	DOUBLE	Tool current [mA]	
tool_temperature	DOUBLE	Tool temperature in degrees Celsius	
tcp_force_scalar	DOUBLE	TCP force scalar [N]	
output_bit_registers0_to_31	UINT32	General purpose bits	
output_bit_registers32_to_63	UINT32	General purpose bits	
output_bit_register_X (X=[64 .. 127])	BOOL	64 general purpose bits X: [64..127] - The upper range of the boolean output registers can be used by external RTDE clients (i.e URCAPS).	3.9.0 / 5.3.0
output_int_register_X (X=[0 .. 47])	INT32	48 general purpose integer registers X: [0..23] - The lower range of the integer output registers is reserved for FieldBus/PLC interface usage. X: [24..47] - The upper range of the integer output registers can be used by external RTDE clients (i.e URCAPS).	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0
output_double_register_X (X=[0 .. 47])	DOUBLE	48 general purpose double registers X: [0..23] - The lower range of the double output registers is reserved for FieldBus/PLC interface usage. X: [24..47] - The upper range of the double output registers can be used by external RTDE clients (i.e URCAPS).	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0
input_bit_registers0_to_31	UINT32	General purpose bits (input read back) This range of the boolean output registers is reserved for FieldBus/PLC interface usage.	3.4.0
input_bit_registers32_to_63	UINT32	General purpose bits (input read back) This range of the boolean output registers is reserved for FieldBus/PLC interface usage.	3.4.0
input_bit_register_X (X=[64 .. 127])	BOOL	64 general purpose bits X: [64..127] - The upper range of the boolean output registers can be used by external RTDE clients (i.e URCAPS).	3.9.0 / 5.3.0
input_int_register_*X* (X=[0 .. 48])	INT32	48 general purpose integer registers X: [0..23] - The lower range of the integer input registers is reserved for FieldBus/PLC interface usage.	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0

input_double_register_*X* (X=[0 .. 48])	DOUBLE	X: [24..47] - The upper range of the integer input registers can be used by external RTDE clients (i.e URCAPS). 48 general purpose double registers X: [0..23] - The lower range of the double input registers is reserved for FieldBus/PLC interface usage. X: [24..47] - The upper range of the double input registers can be used by external RTDE clients (i.e URCAPS).	[0..23] 3.4.0 [24..47] 3.9.0 / 5.3.0
tool_output_mode	UINT8	The current output mode	
tool_digital_output0_mode	UINT8	The current mode of digital output 0	
tool_digital_output1_mode	UINT8	The current mode of digital output 1	
payload	DOUBLE	Payload mass Kg	3.11.0 / 5.5.1
payload_cog	VECTOR3D	Payload Center of Gravity (CoGx, CoGy, CoGz) m	3.11.0 / 5.5.1
payload_inertia	VECTOR6D	Payload inertia matrix elements (Ixx,Iyy,Izz,Ixy,Ixz,Iyz] expressed in kg*m^2	3.15 / 5.11
script_control_line	UINT32	Script line number that is actually in control of the robot given the robot is locked by one of the threads in the script. If no thread is locking the robot this field is set to '0'. Script line number should not be confused with program tree line number displayed on polyscope.	3.14.0 / 5.9.0
ft_raw_wrench	VECTOR6D	Raw force and torque measurement, not compensated for forces and torques caused by the payload	5.9.0

DATA TYPES

Name	Type	Size in bits
BOOL	0 = False, everything else = True	8
UINT8	unsigned integer	8
UINT32	unsigned integer	32
UINT64	unsigned integer	64
INT32	signed integer, two's complement	32
DOUBLE	IEEE 754 floating point	64
VECTOR3D	3xDOUBLE	3x64
VECTOR6D	6xDOUBLE	6x64
VECTOR6INT32	6xINT32	6x32
VECTOR6UINT32	6xUINT32	6x32
STRING	ASCII char array	lengthx8

Network byte order: Data are always sent in network byte order, so big-endian. If you read raw data directly from the socket you will see it in big-endian, and you might need to convert to little. UR's rtde-client library will do the conversion to little.

PROTOCOL

EE = External Executable

CON = Robot Controller

Output: CON -> EE

Input: CON <- EE

HEADER

Field name	Data Type
package size	uint16_t
package type	uint8_t

Summary: All packages use the header.

Supported package types are:

Package Type	Value (DEC)	ASCII equivalent
RTDE_REQUEST_PROTOCOL_VERSION	86	V
RTDE_GET_URCONTROL_VERSION	118	v
RTDE_TEXT_MESSAGE	77	M
RTDE_DATA_PACKAGE	85	U
RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS	79	O
RTDE_CONTROL_PACKAGE_SETUP_INPUTS	73	I
RTDE_CONTROL_PACKAGE_START	83	S
RTDE_CONTROL_PACKAGE_PAUSE	80	P

Direction: Bilateral

Return: Not available

RTDE_REQUEST_PROTOCOL_VERSION

Field name	Data Type
Header	See above
protocol version	uint16_t

Summary: Request the controller to work with "protocol version"

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
accepted	uint8_t

Summary: The controller accepts or not, i.e. either 1 (success) or 0 (failed). On success, the EE should speak the specified protocol version and the CON will answer in that version.

RTDE_GET_URCONTROL_VERSION

Field name	Data Type
Header	See above

Summary: Retrieves the CON major, minor, bugfix and build number.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
major	uint32_t
minor	uint32_t
bugfix	uint32_t
build	uint32_t

Summary: The major, minor, bugfix and build number.

RTDE_TEXT_MESSAGE (PROTOCOL V. 1)

Direction: CON -> EE

Field name	Data Type
Header	See above
message type	uint8_t
message	string

Direction: EE -> CON

Field name	Data Type
Header	See above
message length	uint8_t
message	string
source length	uint8_t
source	string
warning level	uint8_t

Summary: Send an exception, error, warning or info message.

Warning level: EXCEPTION_MESSAGE, ERROR_MESSAGE, WARNING_MESSAGE, INFO_MESSAGE

EE->CON: Exceptions indicate EE program failure without recovery possibilities. Error, warning and info will end up in the PolyScope log.

CON -> EE: Indicates mainly different kinds of protocol failures.

Direction: See above

Return: Not available.

RTDE_TEXT_MESSAGE (PROTOCOL V. 2)

Field name	Data Type
Header	See above
message length	uint8_t
message	string
source length	uint8_t
source	string
warning level	uint8_t

Summary: Send an exception, error, warning or info message.

Warning level: EXCEPTION_MESSAGE, ERROR_MESSAGE, WARNING_MESSAGE, INFO_MESSAGE

EE->CON: Exceptions indicate EE program failure without recovery possibilities. Error, warning and info will end up in the PolyScope log.

CON -> EE: Indicates mainly different kinds of protocol failures.

Direction: Bilateral

Return: Not available.

RTDE_DATA_PACKAGE

Field name	Data Type
Header	See above
recipe id	uint8_t
<variable>	<data type>

Summary: An update to the CON/EE inputs respectively.

The <variable>s are packaged/serialized binary and match the type specified by the SETUP_OUTPUTS or SETUP_INPUTS requests return.

Direction: Bilateral

Return: Not available

RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS (PROTOCOL V. 1)

Field name	Data Type
Header	See above

variable names	string

Summary: Setup the outputs recipe. At the moment the CON only supports one output recipe. The package should contain all desired output variables. The variables names is a list of comma separated variable name strings.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
variable types	string

Summary: Returns the variable types in the same order as they were supplied in the request.

Variable types: VECTOR6D, VECTOR3D, VECTOR6INT32, VECTOR6UINT32, DOUBLE, UINT64, UINT32, INT32, BOOL, UINT8

If a variable is not available, then the variable type will be "NOT_FOUND".

In case of one or more "NOT_FOUND" return values, the recipe is considered invalid and the RTDE will not output this data.

RTDE_CONTROL_PACKAGE_SETUP_OUTPUTS (PROTOCOL V. 2)

Field name	Data Type
Header	See above
output frequency	double
variable names	string

Summary: Setup the outputs recipe. At the moment the CON only supports one output recipe and the output frequency is configurable. For CB-Series robots the frequency must be between 1 and 125 Hz and the output rate will be according to floor (125 / frequency), while for e-Series robots the frequency can go up to 500 Hz. The package should contain all desired output variables. The variable names is a list of comma separated variable name strings.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
output recipe id	uint8_t
variable types	string

Summary: Returns the variable types in the same order as they were supplied in the request.

Variable types: VECTOR6D, VECTOR3D, VECTOR6INT32, VECTOR6UINT32, DOUBLE, UINT64, UINT32, INT32, BOOL, UINT8

If a variable is not available, then the variable type will be "NOT_FOUND".

In case of one or more "NOT_FOUND" return values, the recipe is considered invalid and the RTDE will not output this data (output recipe id = 0).

RTDE_CONTROL_PACKAGE_SETUP_INPUTS

Field name	Data Type
Header	See above

variable names	string

Summary: Setup a CON input recipe.The CON supports 255 different input recipes (0 is reserved). The variables names is a list of comma separated variable name strings.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
input recipe id	uint8_t
variable types	string

Summary: Returns the variable types in the same order as they were supplied in the request.

Variable types: VECTOR6D, VECTOR3D, VECTOR6INT32, VECTOR6UINT32, DOUBLE, UINT64, UINT32, INT32, BOOL, UINT8

If a variable has been claimed by another EE, then the variable type will be "IN_USE".

If a variable is not available, then the variable type will be "NOT_FOUND".

In case of one or more "IN_USE" or "NOT_FOUND" return values, the recipe is considered invalid (input recipe id = 0).

RTDE_CONTROL_PACKAGE_START

Field name	Data Type
Header	See above

Summary: Request the controller to start sending output updates. This will fail if e.g. an output package has not been configured yet.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above
accepted	uint8_t

Summary: The CON accepts or not. Either 1 (success) or 0 (failed).

RTDE_CONTROL_PACKAGE_PAUSE

Field name	Data Type
Header	See above

Summary: Request the CON to pause sending output updates.

Direction: EE -> CON

RETURN

Field name	Data Type
Header	See above

accepted	uint8_t

Summary: The CON will always accept a pause command and return a 1 (success).

RTDE CLIENT PYTHON MODULE

RTDE clients can be implemented in different languages that support socket communication. The purpose of this RTDE client library, written in Python, is to provide an easy starting point and show some example applications. The functionality has been developed for Python 2.7.11.

EXAMPLES

record.py

Use this script as an executable to record output data from the robot and save it to a csv file.

Optional arguments

- –host: name of host or IP address to connect to (default: localhost)
- –port: port number (default: 30004)
- –samples: specific number of samples to record (otherwise the program will record data until receiving SIGINT/Ctrl+C)
- –frequency: the sampling frequency in Herz
- –config: XML configuration file to use - it will use the recipe with key 'out' (default: record_configuration.xml)
- –output: data output file to write to - an existing file will be overwritten (default: robot_data.csv)
- –verbose: enable info logging output to console
- -h: display help

example_plotting.py

Provides a simple way to read and plot the data from a csv file recorded with the *record.py*.

example_control_loop.py

Example of a simple control loop. A configuration with two input recipes and one output recipe is read from XML file and sent to the RTDE server. The control loop consist of a blocking read followed by some very simple data processing before sending new information to the robot.

RTDE MODULE

This section describes the different classes and their public interfaces of the rtde module.

class csv_reader.CSVReader(csvfile, delimiter)

Reads the CSV file and maps each column into an entry in the namespace dictionary of the object. The column header are the dictionary key and the value is an array of data points in the column.

Input parameters:

- csvfile (file): Any file-like object that has a *read()* method.
- delimiter (string): A one-character string used to separate fields. It defaults to ' '.

class csv_writer.CSVWriter(csvfile, names, types, delimiter)

Returns a writer object that can take RTDE DataObjects and convert them into delimited string and write them to a file like object.

Input parameters:

- csvfile (file): Any file-like object that has a *write()* method.
- names (array<string>): list of variable names

- types (array<string>): list of variable types
- delimiter (string): A one-character string used to separate fields. It defaults to ' '.

writeheader()

Write column headers to current line in file based on variable names. Multidimensional variables will get an index appended to the name in each column.

writerow(data_object)

Write a new row to the file based on the provided DataObject.

Input parameters:

- data_object (DataObject): Data object with member variables matching the names of the configured RTDE variables

class rtde_config.ConfigFile(filename)

An RTDE configuration can be loaded from an XML file containing a list of recipes. Each recipe should have a key and a list of field with a variable name and type. An example XML recipe:

```
<?xml version="1.0"?>
<rtde_config>
  <recipe key="out">
    <field name="timestamp" type="DOUBLE"/>
    <field name="target_q" type="VECTOR6D"/>
    <field name="target_qd" type="VECTOR6D"/>
    <field name="speed_scaling" type="DOUBLE"/>
    <field name="output_int_register_0" type="INT32"/>
  </recipe>
  <recipe key="in1">
    <field name="input_int_register_0" type="INT32"/>
    <field name="input_int_register_1" type="INT32"/>
  </recipe>
  <recipe key="in2">
    <field name="input_double_register_0" type="DOUBLE"/>
  </recipe>
</rtde_config>
```

get_recipe(key)

Gets the recipe associated to the specified key given as a list of names and a list of types.

Input parameters:

- key (string): The key associated to the recipe

Return values:

- variables (array<string>): list of variable names
- types (array<string>): list of variable types

class serialize.DataObject():

A data transfer object where the RTDE variable names configured for synchronization has been added to the namespace dictionary of the class for convenient accessing. This means that for example the timestamp can be accessed on an output DataObject like this: *objName.timestamp*. The DataObject is used for both input and output.

recipe_id

The recipe_id is an integer member variable on the DataObject instance used to identify input packages configured in the RTDE server. It is not used for output packages.

class Rtde.RTDE(hostname, port)

The constructor takes a hostname and port number as arguments.

Input parameters:

- hostname (string): hostname or IP of RTDE server
- port (int): [Optional] port number (default value: 30004)

connect()

Initialize RTDE connection to host.

Return value:

- success (boolean)

disconnect()

Close the RTDE connection.

is_connected()

Returns True if the connection is open.

Return value:

- open (boolean)

get_controller_version()

Returns the software version of the robot controller running the RTDE server.

Return values:

- major (int)
- minor (int)
- bugfix (int)
- build (int)

negotiate_protocol_version(protocol)

Negotiate the protocol version with the server. Returns True if the controller supports the specified protocol version. We recommend that you use this to ensure full compatibility between your application and future versions of the robot controller.

Input parameters:

- protocol (int): protocol version number

Return value:

- success (boolean)

send_input_setup(variables, types)

Configure an input package that the external application will send to the robot controller. An input package is a collection of input variables that the external application will provide to the robot controller in a single update. Variables is a list of variable names and should be a subset of the names supported as input by the RTDE interface.The list of types is optional, but if any types are provided it should have the same length as the variables list. The provided types will be matched with the types that the RTDE interface expects and the function returns None if they are not equal. Multiple input packages can be configured. The returned InputObject has a reference to the recipe id which is used to identify the specific input format when sending an update.

Input parameters:

- variables (array<string>): Variable names from the list of possible RTDE inputs
- types (array<string>): [Optional] Types matching the variables

Return value:

- input_data (DataObject): Empty object with member variables matching the names of the configured RTDE variables

send_output_setup(variables, types)

Configure an output package that the robot controller will send to the external application at the control frequency. Variables is a list of variable names and should be a subset of the names supported as output by the RTDE interface. The list of types is optional, but if any types are provided it should have the same length as the variables list. The provided types will be matched with the types that the RTDE interface expects and the function returns False if they are not equal. Only one output package format can be specified and hence no recipe id is used for output.

Input parameters:

- variables (array<string>): Variable names from the list of possible RTDE outputs
- types (array<string>): [Optional] Types matching the variables

Return value:

- success (boolean)

send_start()

Sends a start command to the RTDE server to initiate the actual synchronization. Setup of all inputs and outputs should be done before starting the synchronization.

Return value:

- success (boolean)

send_pause()

Sends a pause command to the RTDE server to pause the synchronization. When paused it is possible to change the input and output configurations and start the synchronization again.

Return value:

- success (boolean)

send(input_data)

Send the contents of a DataObject as input to the RTDE server. Returns True if successful.

Input parameters:

- input_data (DataObject): object with member variables matching the names of the configured RTDE variables

Return value:

- success (boolean)

receive()

Blocking call to receive next output DataObject from RTDE server.

Return value:

- output_data (DataObject): object with member variables matching the names of the configured RTDE variables

DOWNLOAD THE CLIENT FROM THE GITHUB REPOSITORY HERE.