Real-Time Data Exchange (RTDE) Guide - 22229

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

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

Examples are valid for:
CB3 Software version: 3.3
Note that newer software versions may behave differently.

These examples can be used on CB3/CB3.1 from software 3.3

Purpose: Synchronize external executables with the UR controller, for instance URCaps, (https://confluence.ur-update.dk/display/FB/Ethernet+IP+Home) without breaking any real-time properties.

This document first describes the protocol and then provides a Python client reference implementation.

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.

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

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

2. Input: digital and analog outputs and general purpose input registers

The functionality is among others useful for interacting with drivers (e.g. Ethernet/IP), manipulating robot I/O and plotting robot status (e.g. robot trajectories).

Available in SW version 3.3, the RTDE interface is provided when the UR controller is running.

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

The setAfter connecting 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. In return the RTDE interface replies with a list of the variable types or specifies that a specific variable has not been found.

The definition of a synchronization data package format is often referred to as a *recipe.* Each input recipe that has been successfully configured will get a unique recipe id. A list of supported field names and their associated data types can be found in section XX. When setup is complete the data synchronization can be started and paused.

When setup is complete the data synchronization can be started and paused.

After starting the synchronization loop, 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. All the data synchronization uses serialized 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 you can use or modify the provided client sample written in Python.

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, it can request the RTDE interface to speak a specific protocol version. The current protocol version can be found in the example.

Field names and associated types

The following data fields are available through the RTDE interface.

Robot controller inputs

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[A], 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
input_bit_registers32_to_63	UINT32	General purpose bits
input_int_register_*X*	INT32	24 general purpose integer registers X: [0..23]
input_double_register_*X*	DOUBLE	24 general purpose double registers X: [0..23]

Robot controller outputs

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
actual_TCP_force	VECTOR6D	Generalized forces in the TCP
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
actual_digital_input_bits	UINT64	Current state of the digital inputs.
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
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	Digital outputs
runtime_state	UINT32	Program state
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[A], 1=voltage[V]}
standard_analog_input0	DOUBLE	Standard analog input 0 [A or V]
standard_analog_input1	DOUBLE	Standard analog input 1 [A or V]
standard_analog_output0	DOUBLE	Standard analog output 0 [A or V]
standard_analog_output1	DOUBLE	Standard analog output 1 [A or V]
io_current	DOUBLE	I/O current [A]
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 [A]
tool_mode	UINT32	Tool mode
tool_analog_input_types	UINT32	Output domain {0=current[A], 1=voltage[V]} Bits 0-1: tool_analog_input_0 | tool_analog_input_1
tool_analog_input0	DOUBLE	Tool analog input 0 [A or V]

tool_analog_input1	DOUBLE	Tool analog input 1 [A or V]
tool_output_voltage	INT32	Tool output voltage [V]
tool_output_current	DOUBLE	Tool current [A]
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_int_register_*X*	INT32	24 general purpose integer registers X: [0..23]
output_double_register_*X*	DOUBLE	24 general purpose double registers X: [0..23]

Protocol

EE = External Executable

CON = Robot Controller

Output: CON -> EE

Input: CON <- EE

Header

package size	uint16_t
package type	uint8_t

Summary: All packages use the header.

Direction: Bilateral

Return: Not available

RTDE_REQUEST_PROTOCOL_VERSION

Header	See above
protocol version	uint16_t

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

Direction: EE -> CON

Return

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

Header	See above

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

Direction: EE -> CON

Return

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

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

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

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

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_INPUTS

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

Header	See above
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 (recipe id = 0).

RTDE_CONTROL_PACKAGE_START

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

Header	See above
accepted	uint8_t

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

RTDE_CONTROL_PACKAGE_PAUSE

Header	See above

Summary: Request the CON to pause sending output updates.

Direction: EE -> CON

Return

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 developed in a large set of languages with socket communication support. The purpose of the 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)
- --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

This script is an example of a simple way to read and plot the data from a csv file generated using *record.py*.

example_control_loop.py

This script is an 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 is shown below.

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

```
        &lt;?xml version=&quot;1.0&quot;?&gt;
&lt;rtde_config&gt;
    &lt;recipe key=&quot;out&quot;&gt;
        &lt;field name=&quot;timestamp&quot; type=&quot;DOUBLE&quot;/&gt;
        &lt;field name=&quot;target_q&quot; type=&quot;VECTOR6D&quot;/&gt;
        &lt;field name=&quot;target_qd&quot; type=&quot;VECTOR6D&quot;/&gt;
        &lt;field name=&quot;speed_scaling&quot; type=&quot;DOUBLE&quot;/&gt;
        &lt;field name=&quot;output_int_register_0&quot; type=&quot;INT32&quot;/&gt;
    &lt;/recipe&gt;
    &lt;recipe key=&quot;in1&quot;&gt;
        &lt;field name=&quot;input_int_register_0&quot; type=&quot;INT32&quot;/&gt;
        &lt;field name=&quot;input_int_register_1&quot; type=&quot;INT32&quot;/&gt;
    &lt;/recipe&gt;
    &lt;recipe key=&quot;in2&quot;&gt;
        &lt;field name=&quot;input_double_register_0&quot; type=&quot;DOUBLE&quot;/&gt;
    &lt;/recipe&gt;
&lt;/rtde_config&gt;
```

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