ROCKETFARM

OPC UA

ROCKETFARM OPC UA

USER MANUAL

URCap Version 3.0

Language:

ENGLISH

Enabling OPC UA for UNIVERSAL ROBOTS

1 - Document Revision Information

Date	Description	Author
01.10.2018	Initial Issue	CM,LÅ,AL
12.10.2018	Update	BR
07.11.2018	Updated content for version 1.0.1	AL
13.11.2018	Updated content for version 1.0.2	AL
15.11.2018	Added requirements	AL
03.12.2018	Updated content for version 1.0.3	AL
10.03.2020	Updated content for version 2.0 beta	CM
11.05.2020	Updated screenshots and content for version 2.0 RC3	CM
21.05.2020	Updated content for version 2.0	CM
29.04.2022	Instant evaluation license option added	CM
15.09.2022	OPC VDMA Robotics added	CM

2 - License

The OPC UA URCap is based on the open62541 open source OPC UA implementation, which is licensed under the Mozilla Public License V2.0

Disclaimer

The information contained herein is the property of Rocketfarm AS and shall not be reproduced in whole or in part without prior written approval of Rocketfram AS. The information herein is subject to change without notice and should not be construed as a commitment by Rockerfarm AS. Rocketfarm AS assumes no responsibility for any errors or omissions in this document.

The Rocketfarm logo is a registered trademark of Rocketfarm AS.

Table of Contents

3 - Introduction

The OPC UA URCap is a software module that extends the functionality of the Universal Robots programming interface with basic data exchange via the industry standard OPC UA protocol.

The functions provided by this URCap are *application-level* meaning that the OPC UA data exchange has to be controlled by the main robot program.

The functions of the OPC UA URCap are further divided into OPC UA client and OPC UA server. Use the client functions to connect to any remote OPC UA server and exchange data with the remote server. Use the server functions to provide data to remote OPC UA clients. Both client and server functions can be used simultaneously.

The typical use cases of this URCap are receiving commands to be performed by the robot, or reporting the progress to a higher level supervisor system. The URCap is not made for controlling the robot in real time.

The URCap conforms to the OPC VDMA 40010 Robotics Companion Specification v1.0. Objects and variables of the OPC Robotics Information Model are automatically published when the server is enabled. See https://reference.opcfoundation.org/Robotics/docs/ for further information about the OPC Robotics Information model.

4 - Installation

4.1 - Requirements

The OPC UA URCap is developed using functionality available in the following URCap API version 1.5.

- CB3.1: available in Universal Robots software v.3.8 or greater,
- E-series: available in Universal Robots software v.5.2 or greater.

It is recommended to enable the network interface on the robot and configure the TCP/IP settings properly before installing this URCap.

4.2 - Installing the URCap

Install the OPC UA URCap as any other URCaps.

- Go to Setup Robot
- Select URCaps
- press the + button.

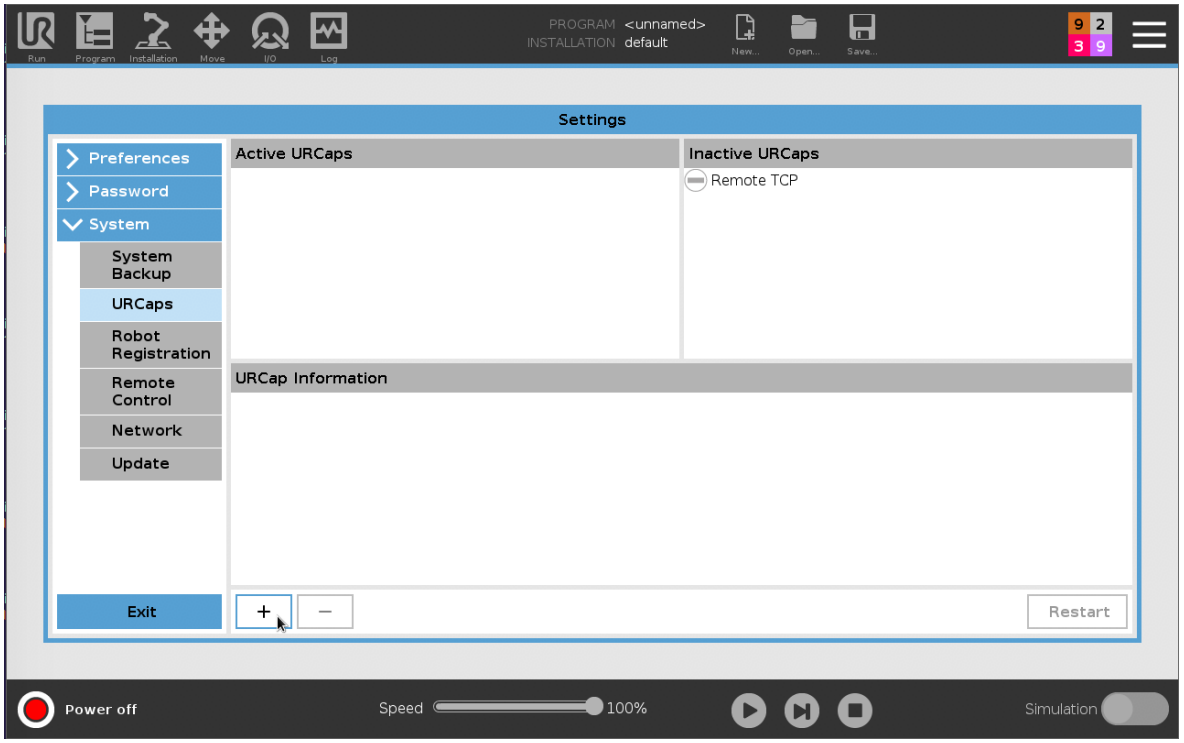

Figure 1 The URCaps setup screen

Select the OPC UA URCap installer package, and press Open.

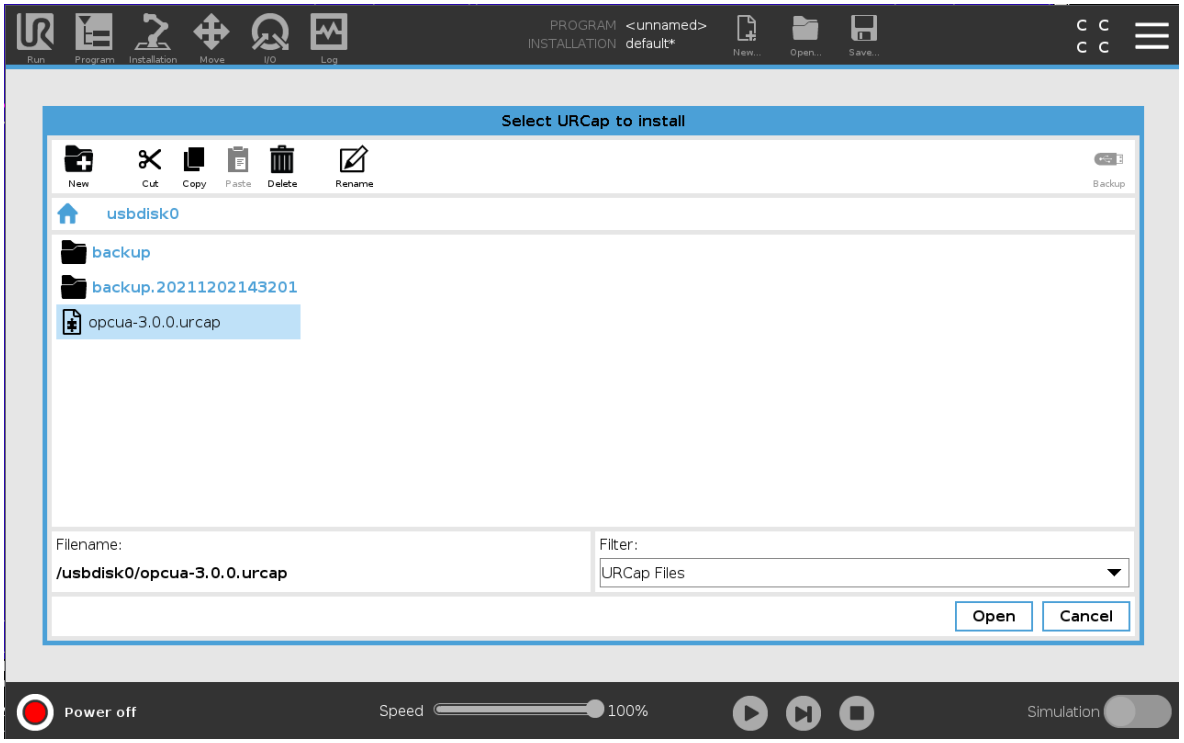

Figure 2 Select the URCap to be installed

Press the Restart button when asked.

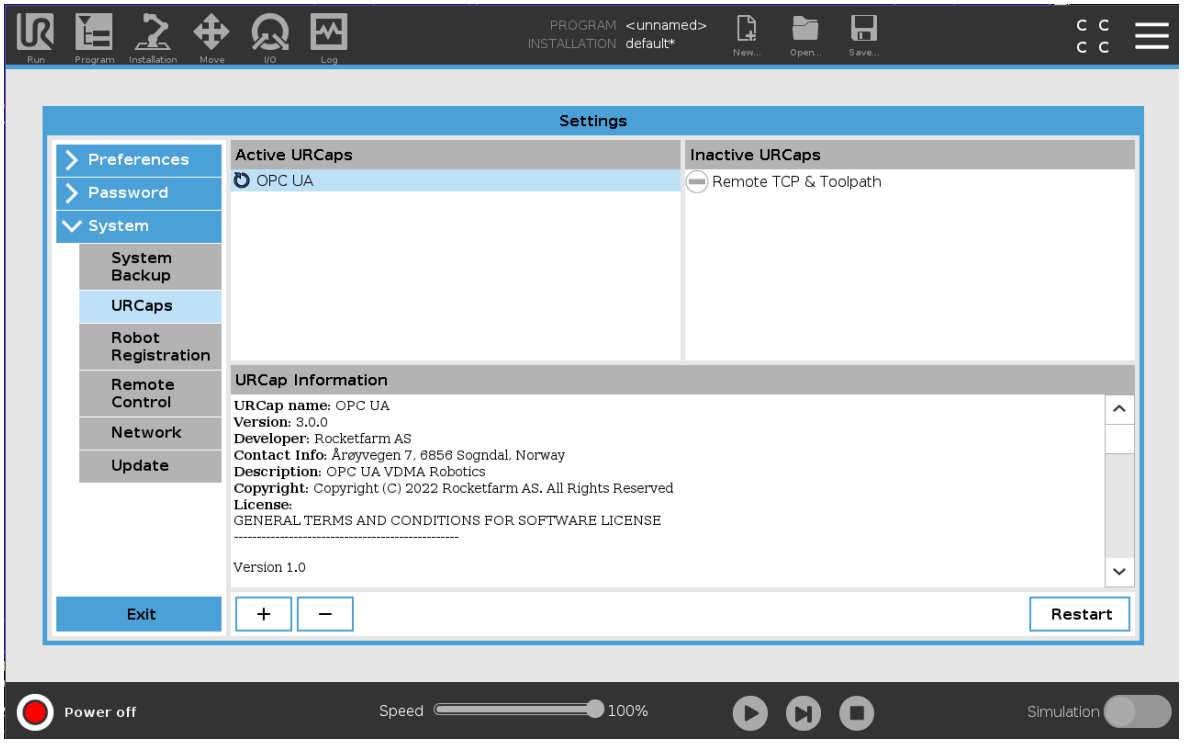

Figure 3 A restart is required after installing the URCap

After restarting the robot, the OPC UA URCap is ready. This is indicated by a green check mark at the URCap name.

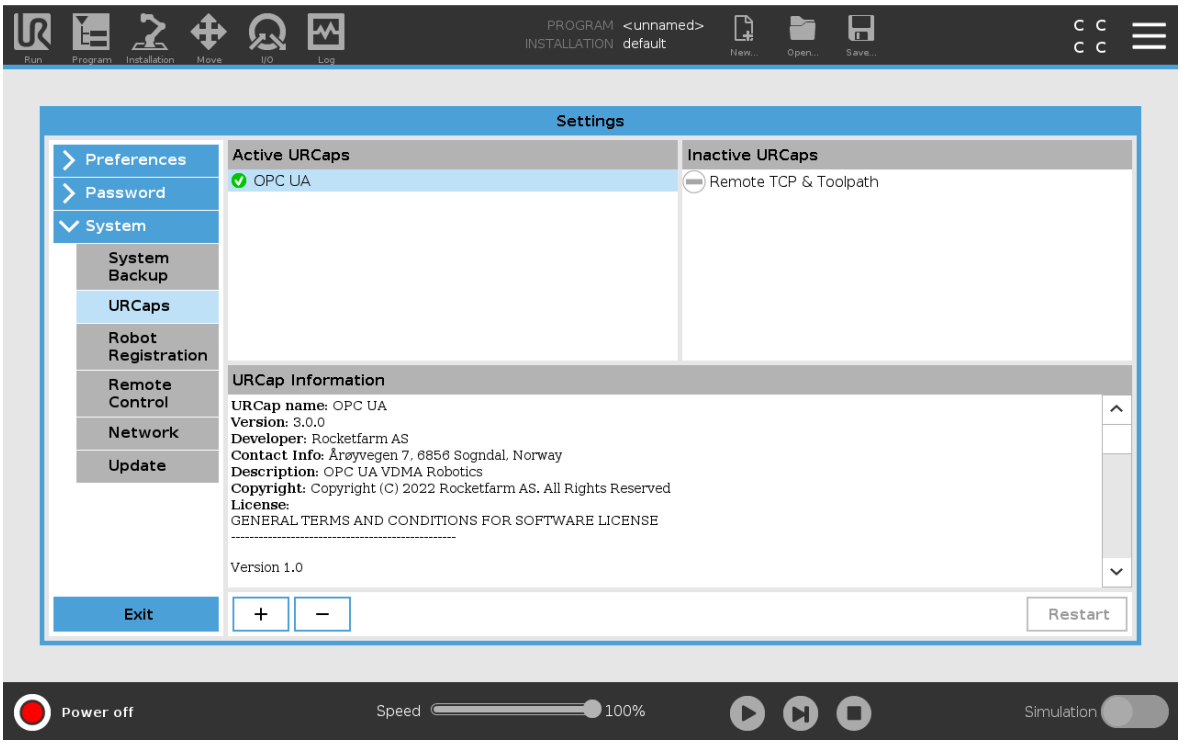

Figure 4 The green check mark has to be visible at the URCap name

5 - Configuration

The OPC UA URCap can be configured in the OPC UA section of the Installation page. The configuration is stored along with the robot installation.

There are separate configuration pages for the clients, the local server, the underlying daemon process, and the license.

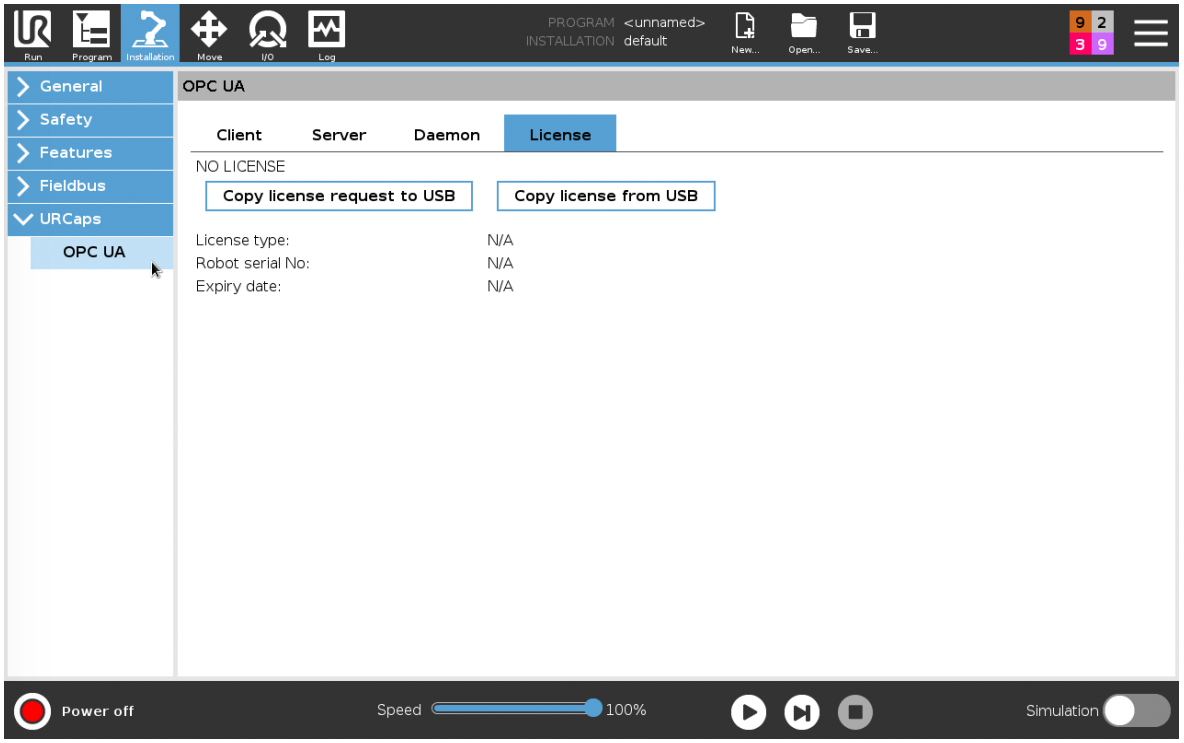

Figure 5 The client, server, daemon, and license configuration pages

5.1 -License - Request and Installation

To use the OPC UA URCap, you will need to obtain a valid license file for the robot in question.

An evaluation license can be activated locally, and will enable all functions to work for 30 days. A permanent license is always bound to a specific robot with its serial number.

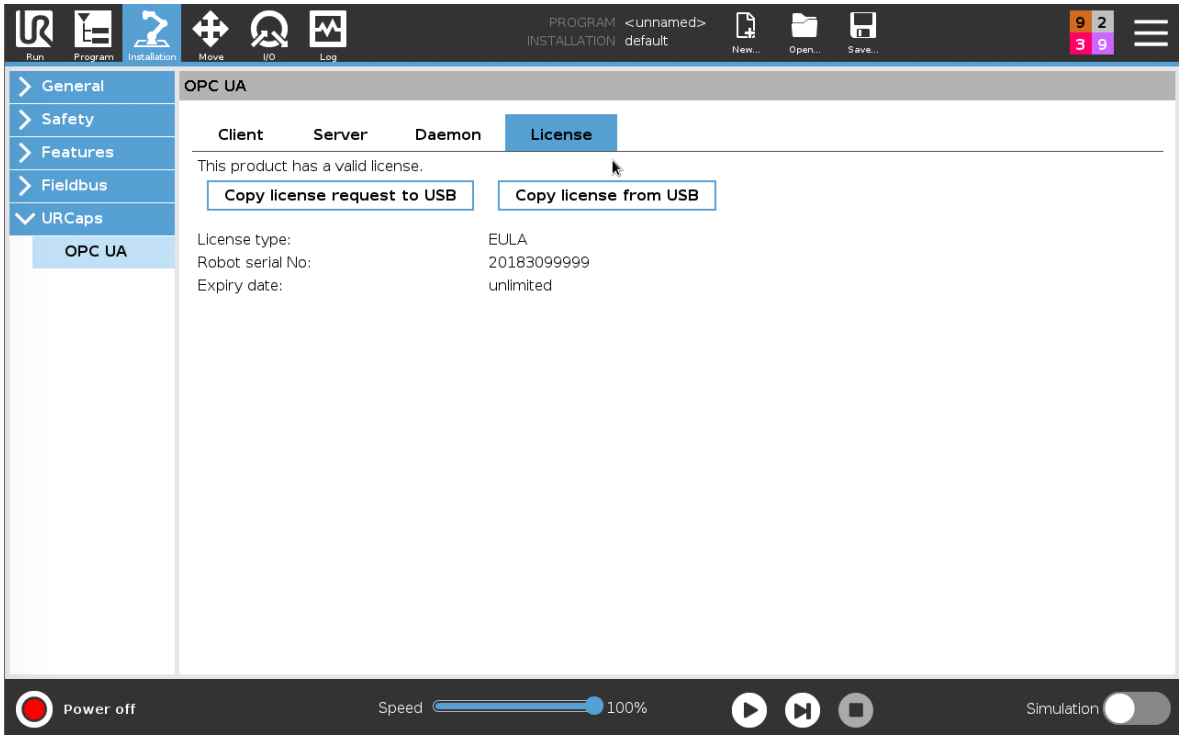

Figure 6 The license page

5.1.1 - Requesting a license

Permanent licenses require a unique license request that is generated on the robot. Perform the following steps to get a valid license request on a USB disk:

- Insert a USB drive into the robot
- Click the “Copy license request to USB” button in the OPC UA URCap
- Disconnect the USB drive

Send the request file to license@rocketfarm.no. Rocketfarm will return a valid license file for this specific robot.

5.1.2 - Installing a license file

After receiving a license file from Rocketfarm, perform the following steps:

- Copy the license file to the root folder on a USB drive
- Connect the USB drive to the robot
- Click the “Copy license from USB” button, in the OPC UA URCap
- Disconnect the USB drive

The OPC UA “Enable daemon” button should now be available to activate and use the URCap.

5.1.3 - Activating an evaluation license

An instant evaluation license can be activated when pressing the button “Copy license request to USB” and then answering "Yes" to the question that appears. The evaluation license will enable all functions for 30 days.

Note: An evaluation license cannot be upgraded to another evaluation license after expiration.

5.2 - Activating and deactivating the OPC UA

The OPC UA URCap can be activated and deactivated by pressing the "Enable daemon" and "Disable daemon" buttons respectively. When the URCap is deactivated, no OPC UA related script code will be generated, and the OPC UA daemon is not running.

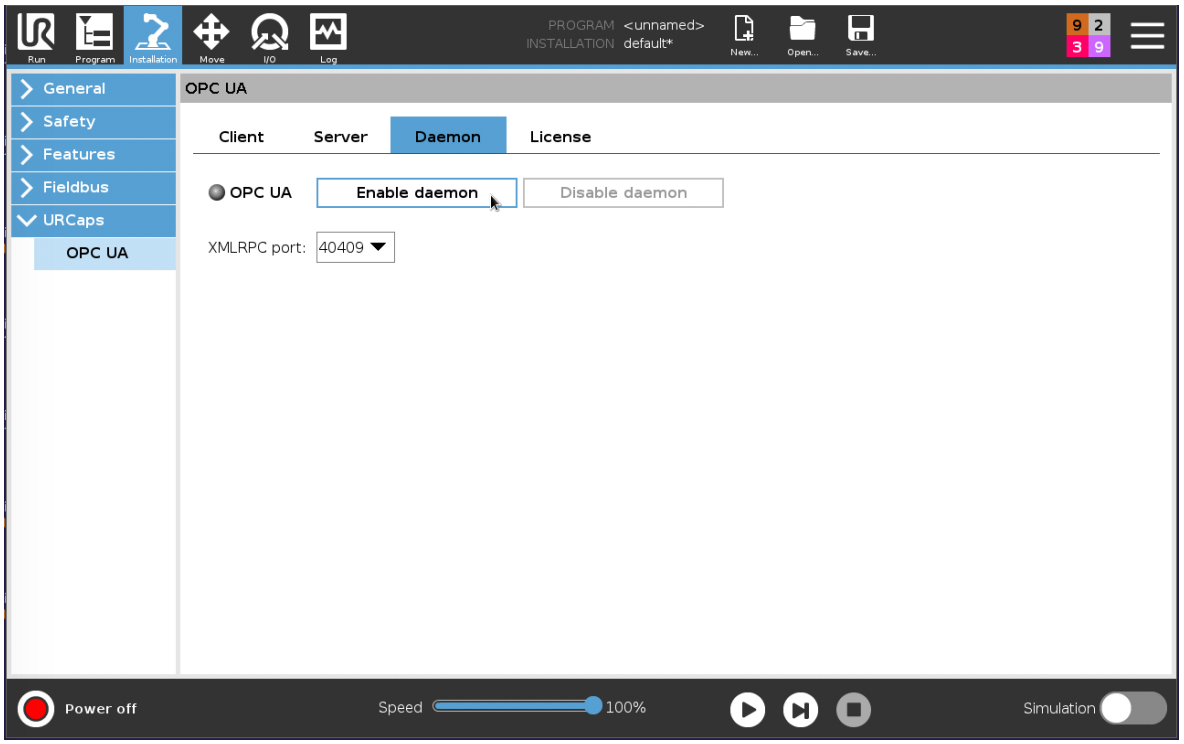

Figure 7 Enable and disable the OPC UA URCap

5.3 - Creating a connection to a remote server

When the OPC UA URCap is active,go to the client configuration page by selecting the "Client" tab.

Click on "Add new" to create a new client with empty configuration.

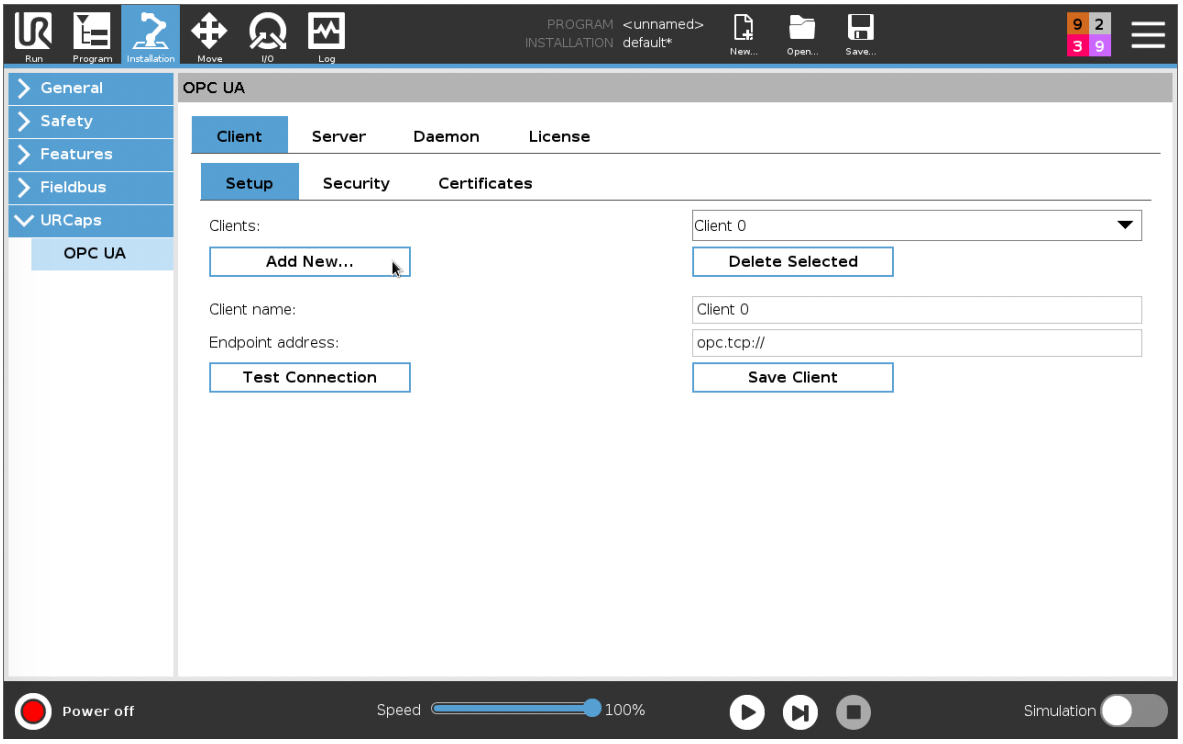

Figure 8 Add a new client connection

Give the connection a friendly name that is easy to read and understand. This will be used later in the robot program.

Enter the address of the remote OPC UA server you want to connect to. Usually this looks like `opc.tcp://host:4840` where host is either the IP address or the host name of the server. Refer to the user manual of your OPC UA server or ask the system administrator for further details.

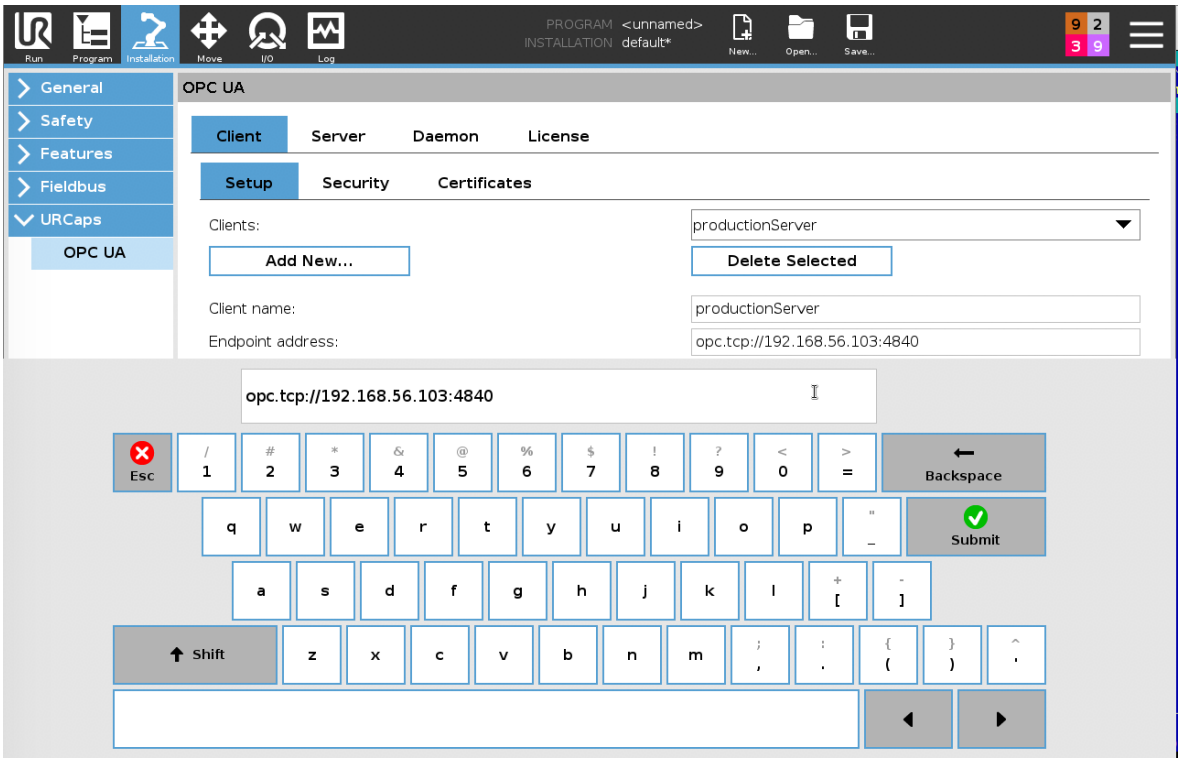

Figure 9 Client name and endpoint address configuration

Security settings for the client can be configured on the Security tab. Choose the security policy, message security mode, and the user authentication policy as required by the remote server.

The available message security modes are:

- None
- Sign
- Sign and encrypt

The available security policies are:

- None (not selectable, used when message security mode is None)
- Basic128Rsa15 (deprecated and should not be used)
- Basic256 (deprecated and should not be used)
- Basic256Sha256

The available user authentication policies are:

- Anonymous
- Username with password

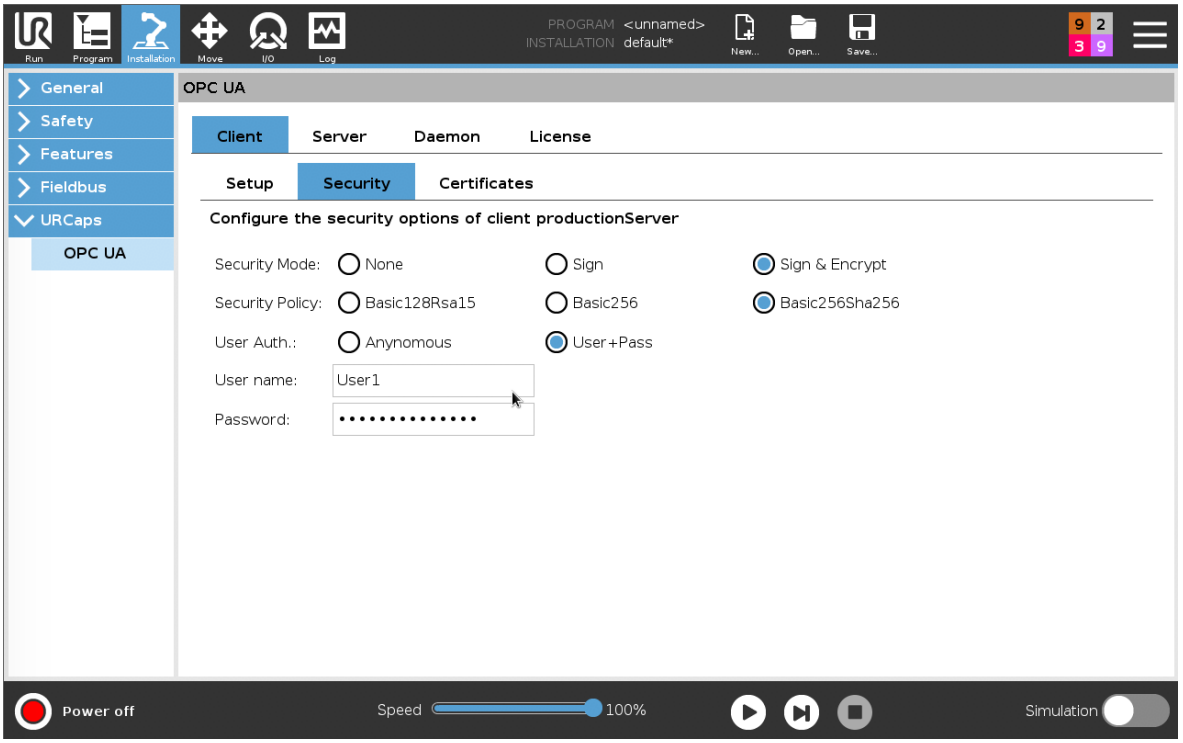

Figure 10 Security options of a client connection

Secure client connections require that you install the appropriate certificate and primary key on the Certificates tab. Both files should be in the DER format, with the following file name conventions:

- certificate: client.crt.der
- private key: client.key.der

Insert an USB disk containing both files in the root folder, and press the "Copy from USB" button. The information lines above the buttons should display information about the newly installed files.

Note: A self-signed certificate will be generated automatically if no certificate is installed.

You can delete the installed certificate and primary key later by pressing the "Delete all" button.

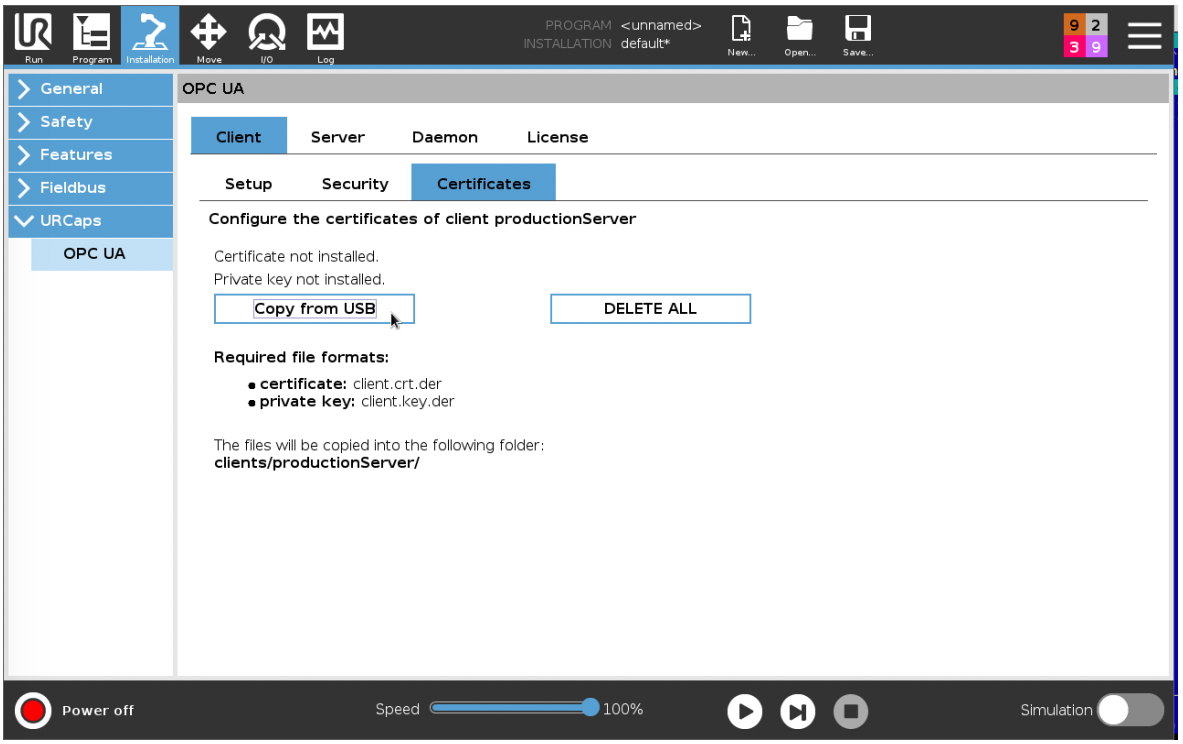

Figure 11 Install client certificate and private key

When the security options are configured, return to the "Setup" tab and press the "Test connection" button. The OPC UA URCap will try to connect to the server and discover the available variables.

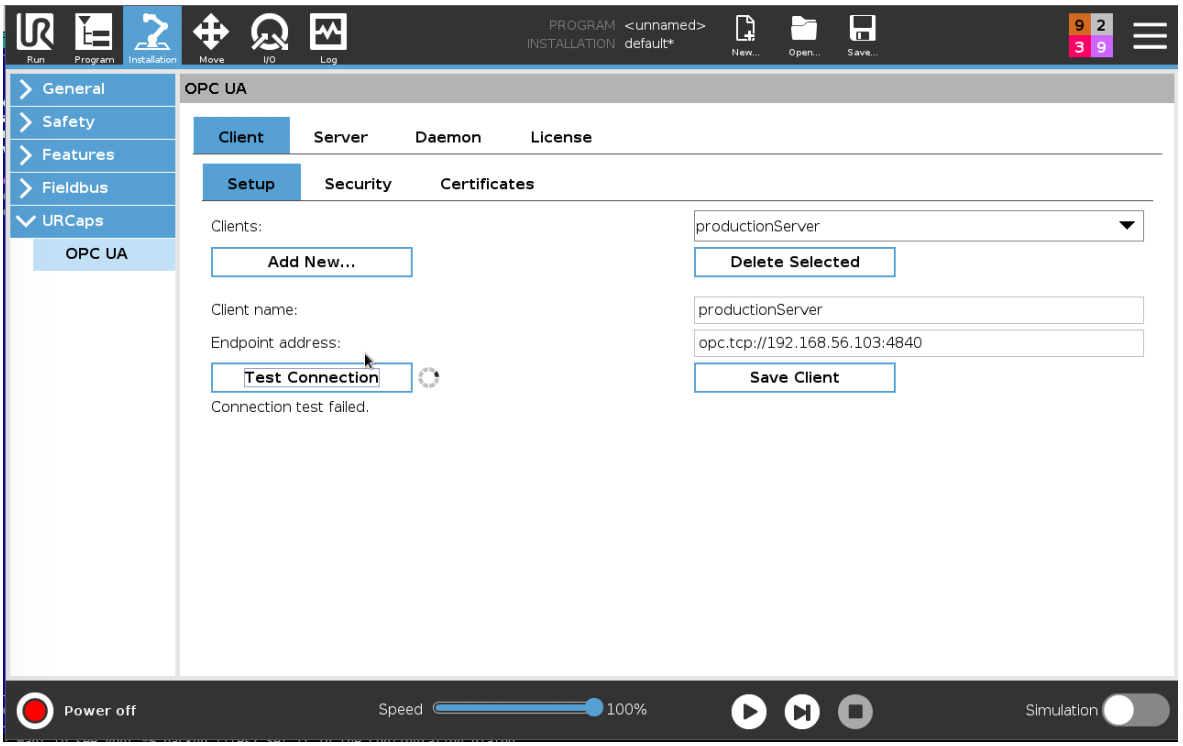

Figure 12 Test the client connection after its configuration is completed

If everything goes well, the "Connection test OK" message appears. In this case, press the "Save Client" button.

In case you see the error message "Connection test failed" check that you have entered the server address properly and selected the required security options, and that the OPC UA server is running and is accessible from the robot via network.

5.4 - Creating a local server

The OPC UA URCap includes a local OPC UA server that other clients can access. In order to get the local server running, you need to specify the TCP port the server can listen on, and create at least one OPC UA variable. Optionally, the security settings can be configured.

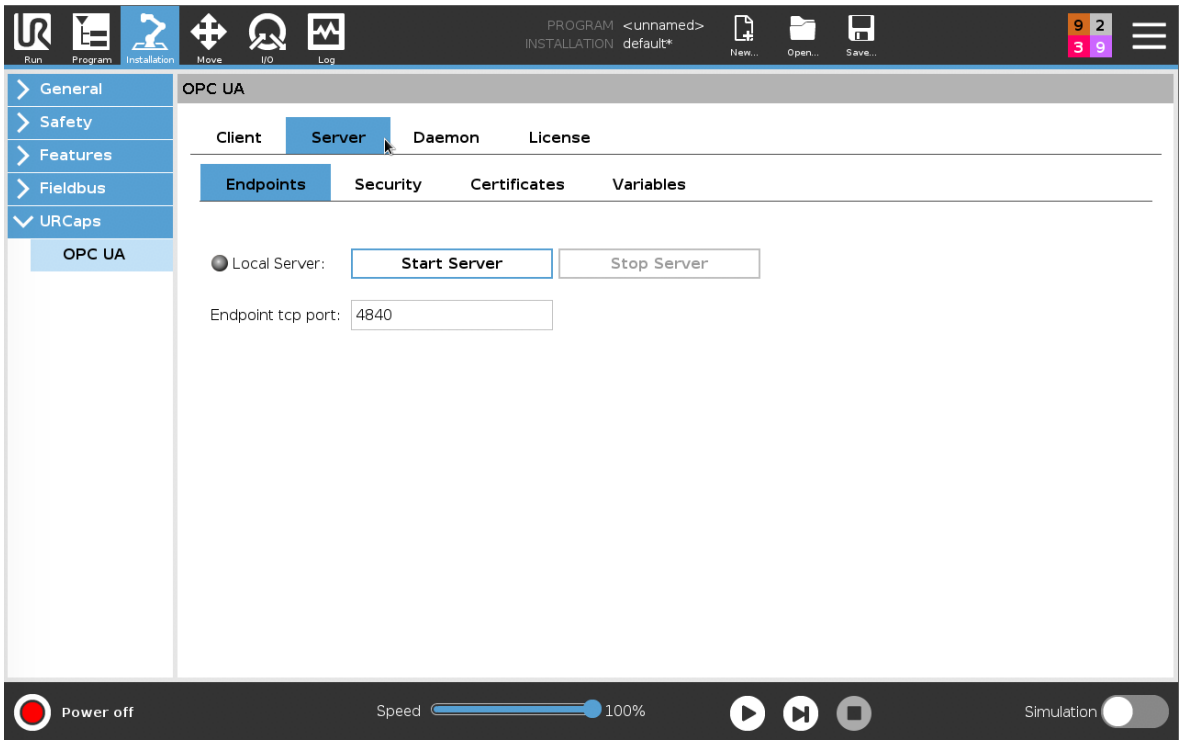

Figure 13 choose a TCP port and start the local OPC UA server

The server can be configured to accept clients with different security settings. Choose the allowed message security modes, security policies, and user authentication policies.

The available message security modes are:

- None
- Sign
- Sign and encrypt

The available security policies are:

- None (not selectable, used when message security mode is None)
- Basic128Rsa15 (deprecated and should not be used)
- Basic256 (deprecated and should not be used)
- Basic256Sha256

The available user authentication policies are:

- Anonymous
- Username with password

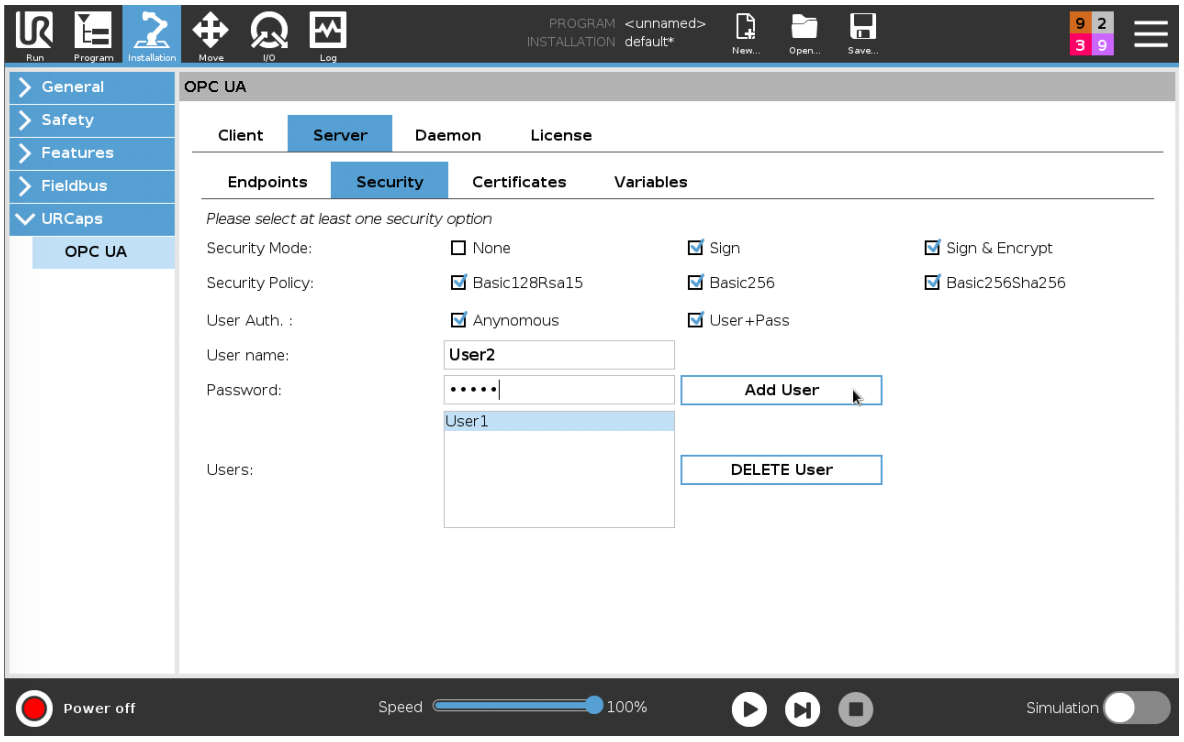

Figure 14 Server security options

Security mode requires that you install the appropriate certificate and primary key on the Certificates tab. Both files should be in the DER format, with the following file name conventions:

- certificate: server.crt.der
- private key: server.key.der

Insert an USB disk containing both files in the root folder, and press the "Copy from USB" button. The information lines above the buttons should display information about the newly installed files.

Note: A self-signed certificate will be generated automatically if no certificate is installed.

You can delete the installed certificate and primary key later by pressing the "Delete all" button.

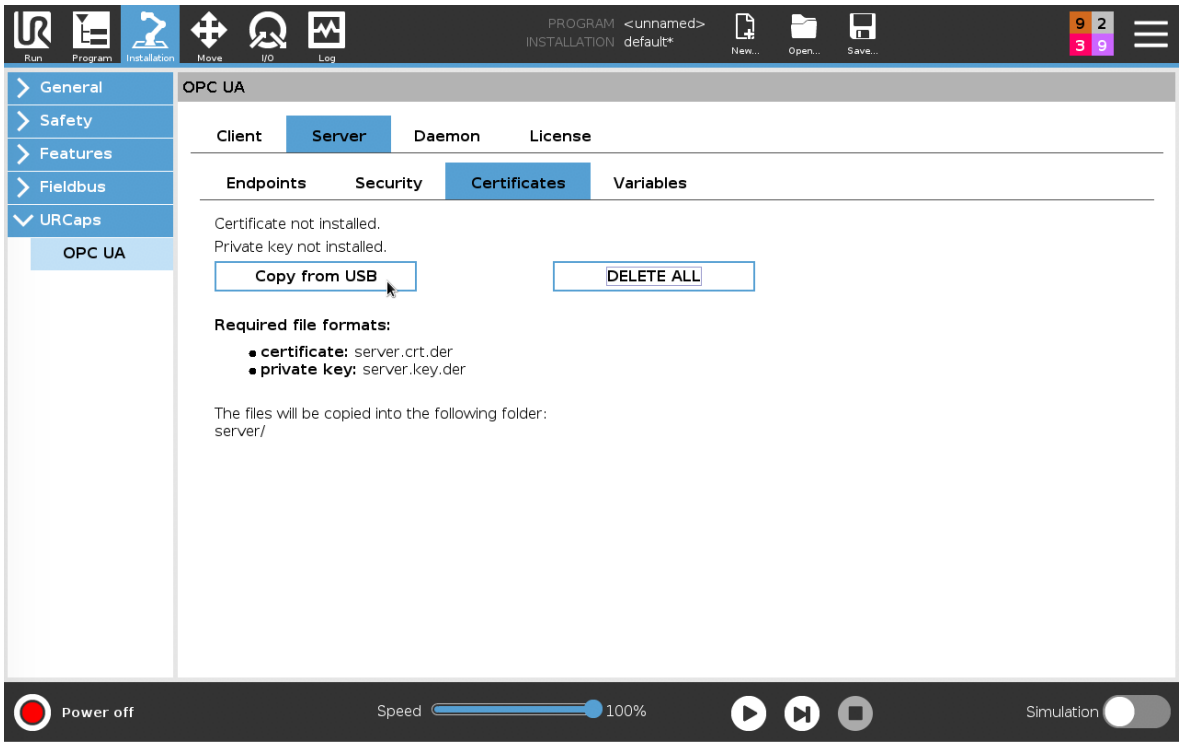

Figure 15 Install server certificate and private key

5.4.1 - Initializing the server from the installation

Go to the "Variables" tab and uncheck the "Initialize from file" checkbox. Enter the name, select the variable type, and enter the default value of the variable. Press the "Add Variable" button.

Currently the following data types are supported:

- Boolean: logical value (true or false)
- Integer: integer value

- Double: double precision floating point value
- String: text

To delete one or more existing variables, select the variable(s) in the variable list, and press the "Delete Variable" button.

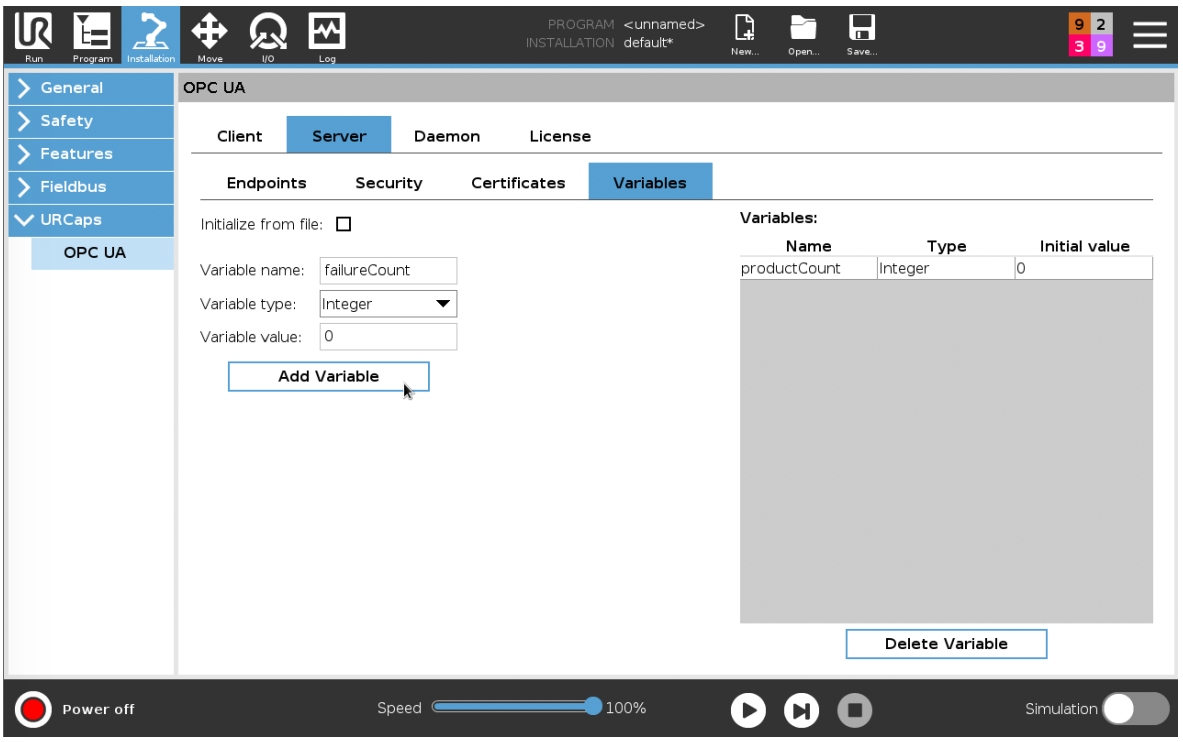

Figure 16 The server variables editor

Note: The server cannot be started until at least one OPC UA variable is defined.

5.4.2 - Initializing the server from external INI-file (advanced users)

The OPC UA server can be initialized from an external INI-file.

Make sure you have copied the ini-file into the "`/programs`" directory on the robot before starting the OPC server.

Select the checkbox "Initialize from file" and enter the file name (e.g. opcua.ini) that contains the variable definitions.

Each line in the ini-file defines one and only one OPC UA variable. The format of one line is the following:

name;type;default

- name: the name of the OPC UA variable
- type: type of the variable (bool, int, double, string)
- default: default value of the OPC UA variable on server start.

If the file format is correct, the defined variables are shown in the "Variables" window.

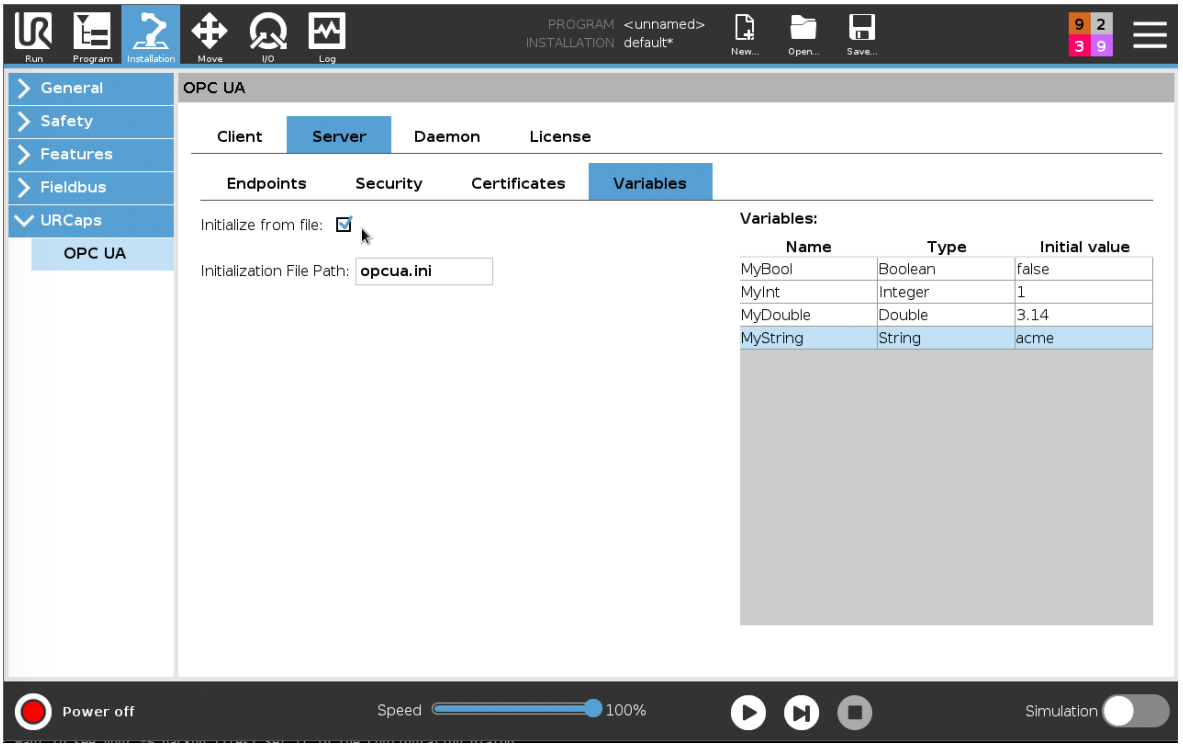

Figure 17 Initialize the OPC UA server from INI-file

5.4.3 - Starting and stopping the server

Press the "Start server" button to start the local OPC UA server.

Press the "Stop server" button to stop the local OPC UA server.

This setting will be saved along with the robot installation, so the server can start automatically on the next startup

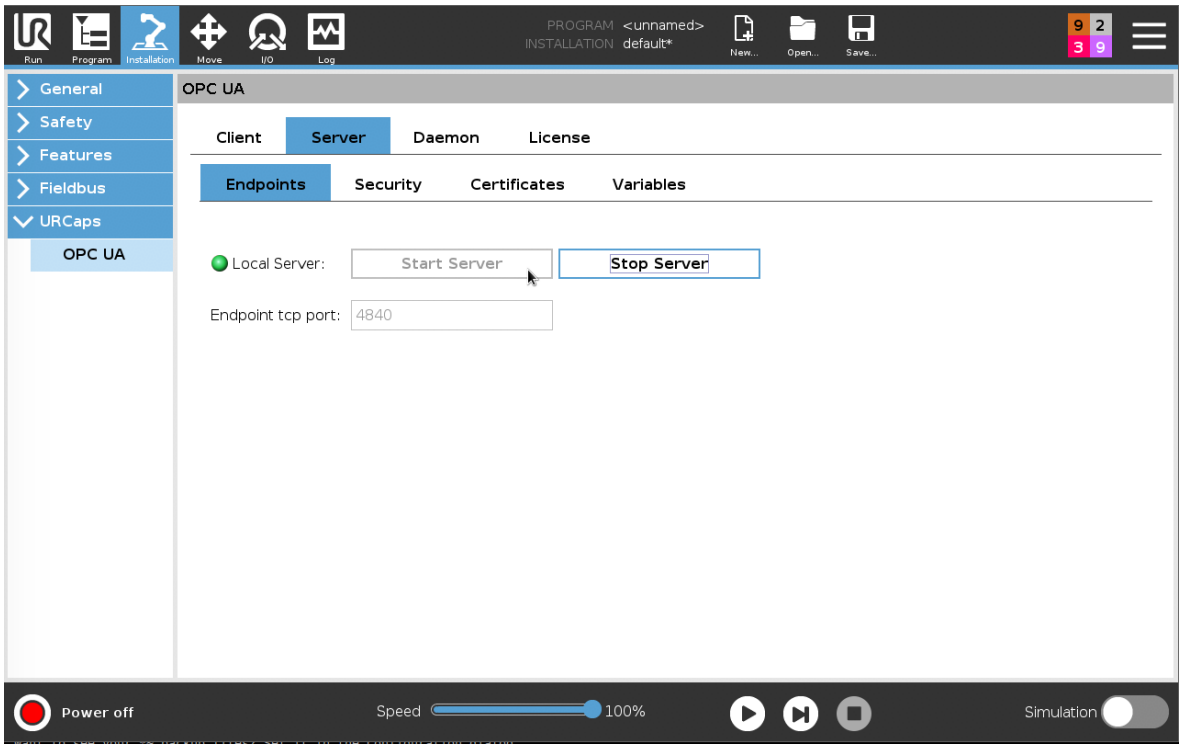

Figure 18 Starting and stopping the local OPC UA server

6 - Programming

The functions of the OPC UA URCap can be accessed in 3 different ways:

- Program nodes
- Script functions
- Low-level API (the OPC UA daemon)

6.1 - Program nodes

The easiest and most convenient way of using the OPC UA URCap is the built-in program nodes, although the functionality is limited.

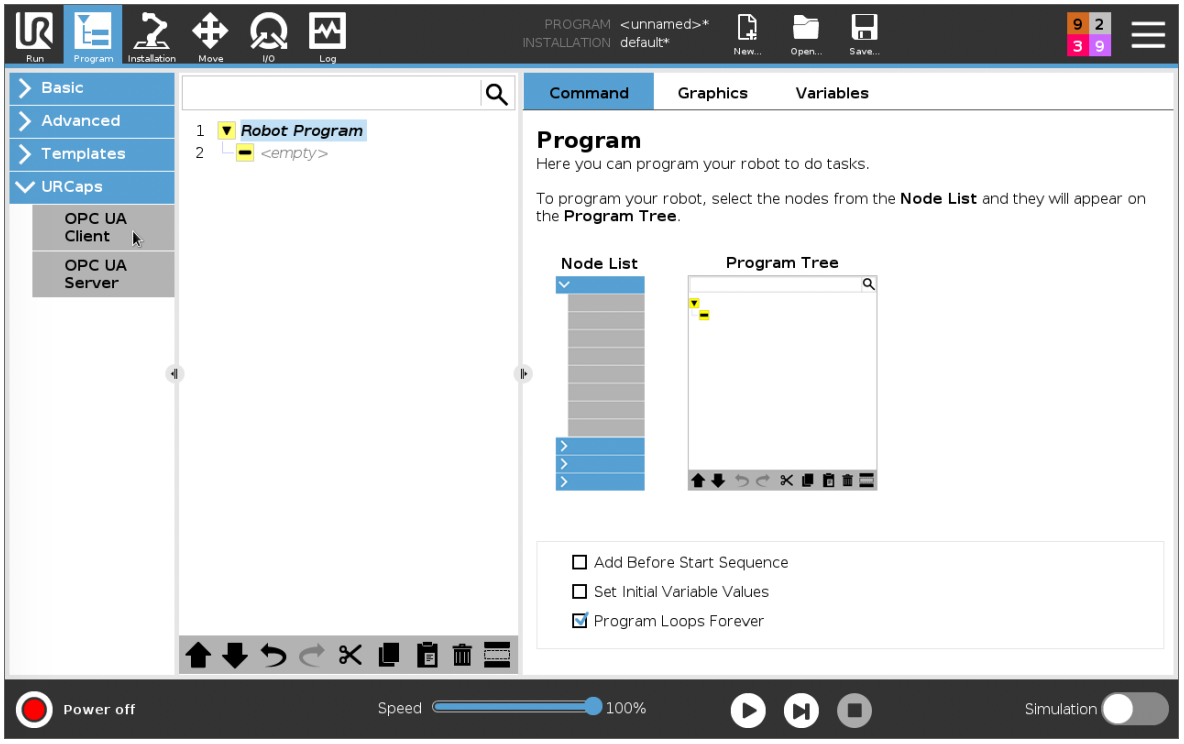

Figure 19 The OPC UA program nodes in the program editor

6.1.1 - Client

Select the "OPC UA Client" program node to communicate with a remote OPC UA server.

The drop-down list shows the *friendly name* of each remote server connection, as they were configured on the Installation screen. Select the server that you want to connect to.

Choose the operation you want to perform:

- connect: open the connection to the remote server
- read: copy the value of a remote OPC UA variable into a URScript program variable
- write: copy the value of a URScript program variable into a remote OPC UA variable
- disconnect: close the connection to the remote server

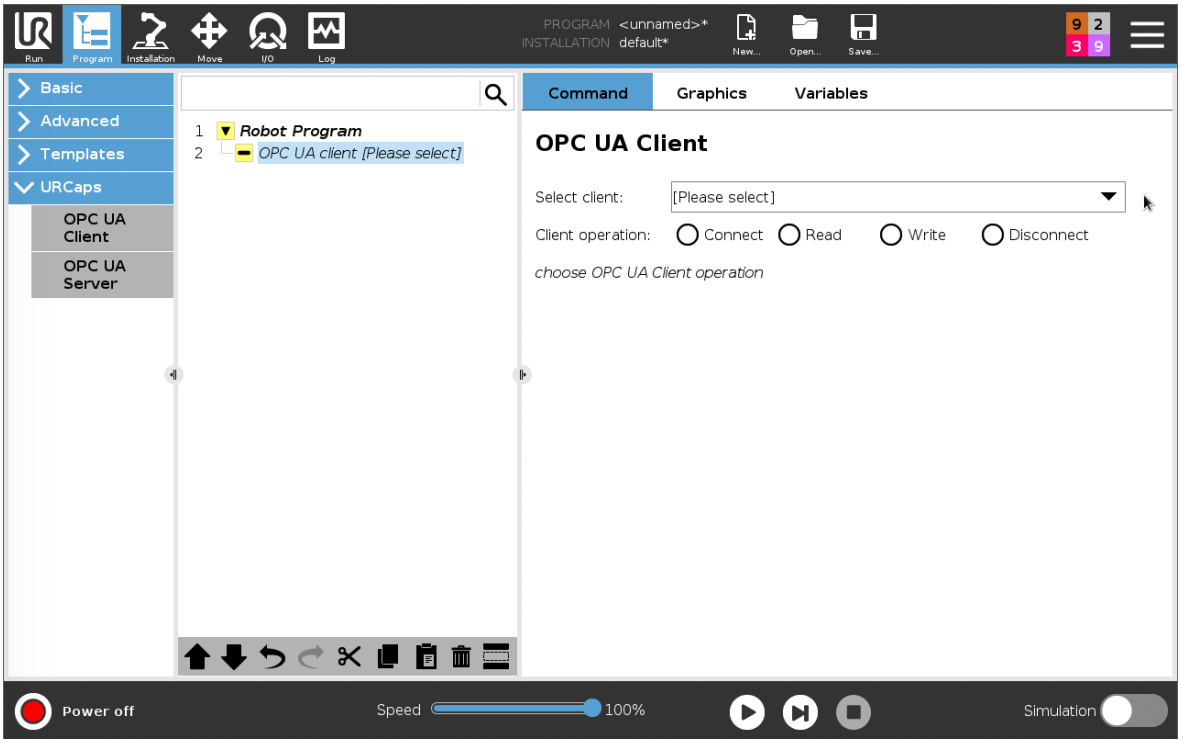

Figure 20 OPC UA client program node

6.1.1.1 - Connect

Select the "Connect" operation to connect to a remote OPC UA server. Choose a client that has been configured during the Installation.

Select the action to be performed when the connection to the remote server fails. The following options are available:

- Do nothing: the robot program can continue without any action;
- Retry until success: the robot program keeps trying to connect;
- Terminate program: the robot program stops with an error message.

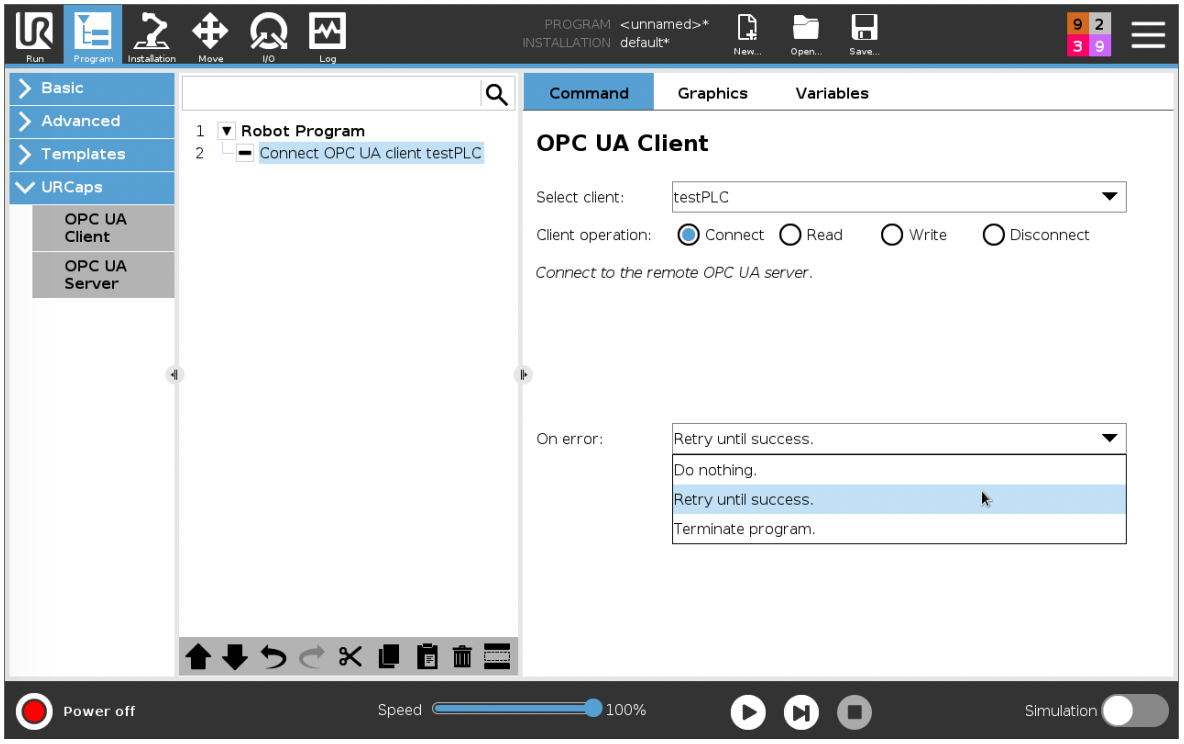

Figure 21 Client connect operation

6.1.1.2 - Read

Select the "Read" operation to copy the value of a remote OPC UA variable into the UR program variable.

The data structure on the remote server is shown in a tree view. Use the "Reload variables" button to refresh the data structure. This can be necessary when the server structure has changed since it was configured in the Installation.

Select the appropriate row and click "Confirm Selection". The data type of the selected value is displayed below the tree.

Choose an existing UR program variable, or enter the name of a new variable and press "Create new".

Select the action to be performed when the read operation fails.

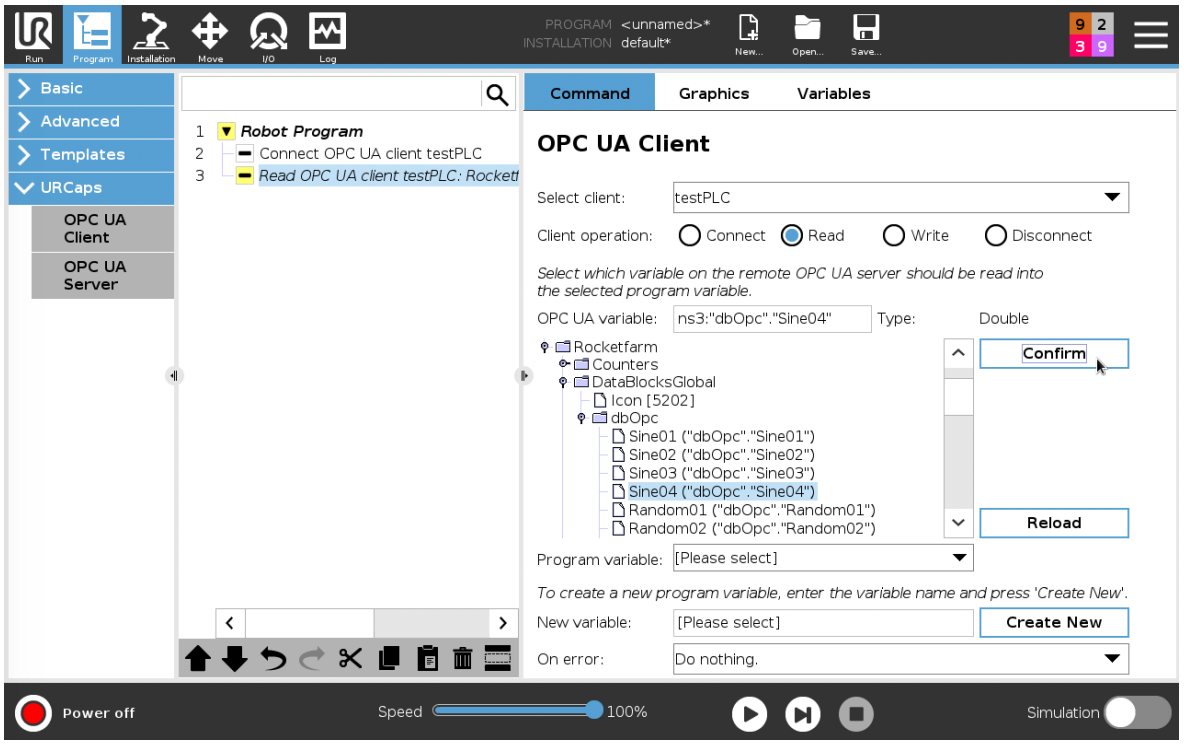

Figure 22 Client read operation

6.1.1.3 - Write

Select the "Write" operation to copy the value of a UR program variable into a remote OPC UA variable.

The data structure on the remote server is shown in a tree view. Use the "Reload variables" button to refresh the data structure. This can be necessary when the server structure has changed since it was configured in the Installation.

Select the appropriate row and click "Confirm Selection". The data type of the selected value is displayed below the tree.

Choose an existing UR program variable whose value will be written to the remote server.

Select the action to be performed when the write operation fails.

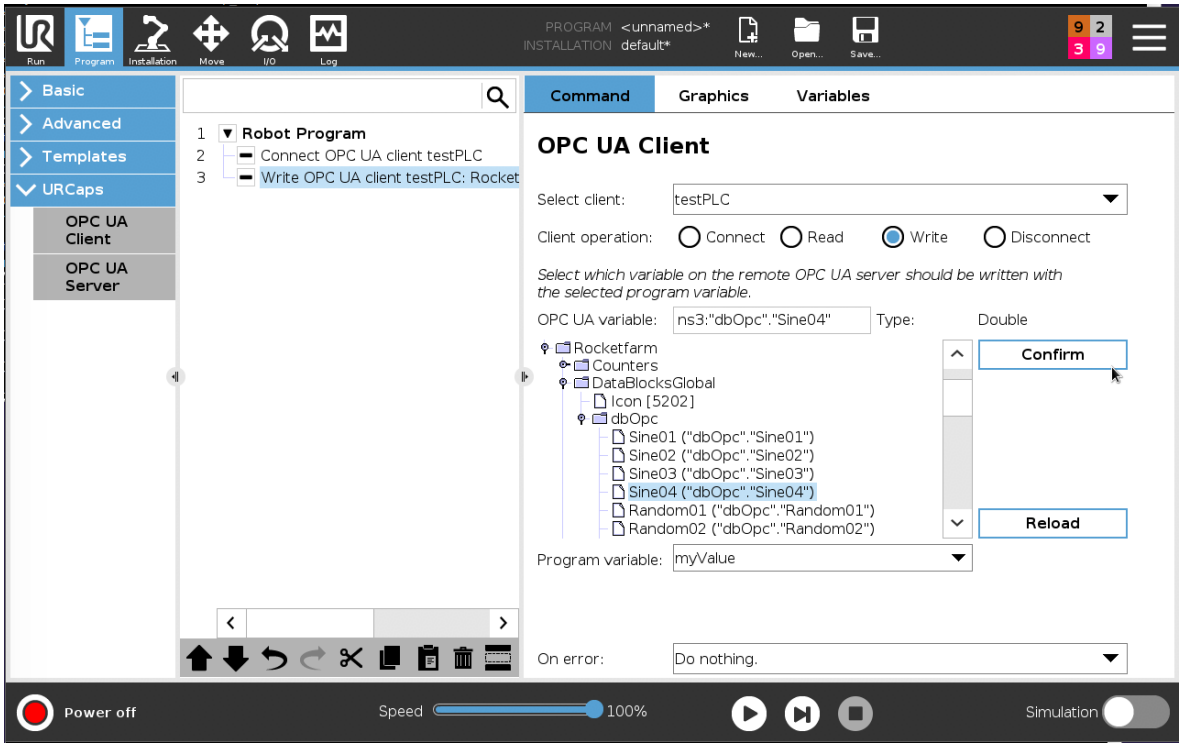

Figure 23 Client write operation

6.1.1.4 - Disconnect

Select the "Disconnect" operation to close the connection to a remote OPC UA server. Choose a client that has been configured during the Installation.

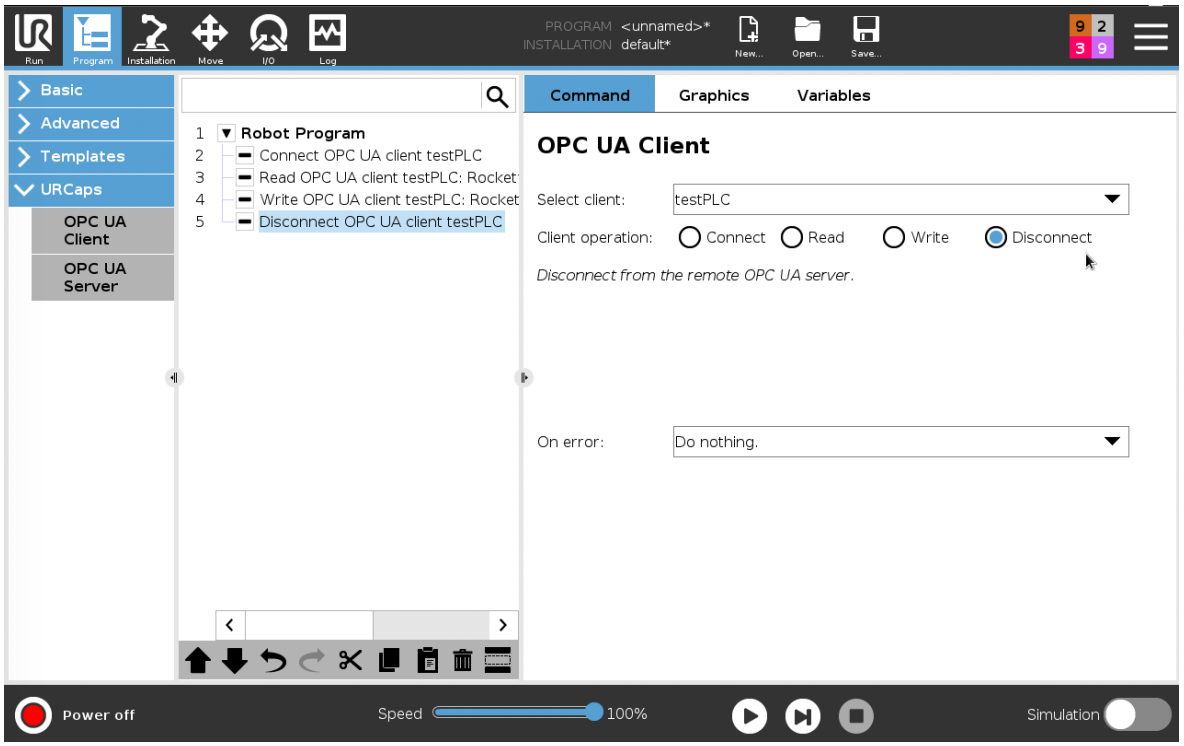

Figure 24 Client disconnect operation

6.1.2 - Server

Use this program node if you want to exchange data between the local OPC UA server and the main robot program.

6.1.2.1 - Read

Select the "Read" operation to copy the value of a local OPC UA variable into the UR program variable.

Select the appropriate OPC UA variable from the list. The data type of the selected value is displayed below the list.

Choose an existing UR program variable, or enter the name of a new variable and press "Create new".

Select the action to be performed when the read operation fails.

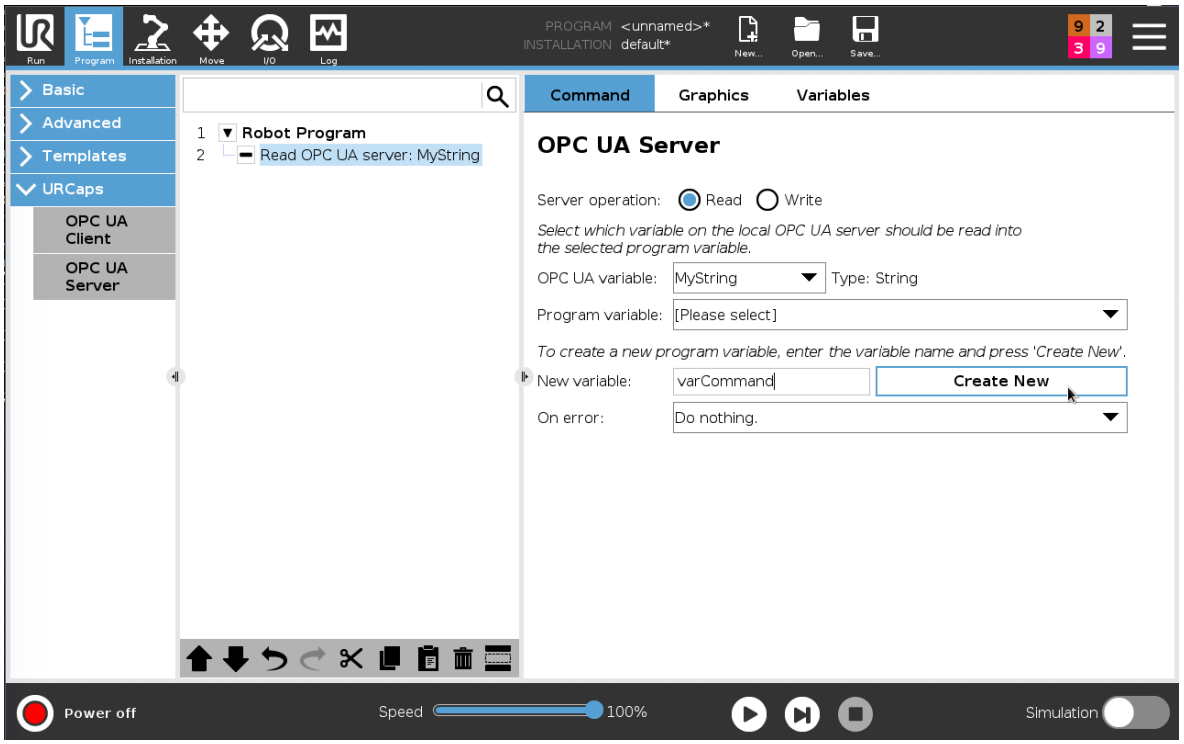

Figure 25 Server read operation

6.1.2.2 - Write

Select the "Write" operation to copy the value of a UR program variable into a local OPC UA variable.

The data structure on the remote server is shown in a tree view. Use the "Reload variables" button to refresh the data structure. This can be necessary when the server structure has changed since it was configured in the Installation.

Select the appropriate OPC UA variable from the list. The data type of the selected value is displayed below the list.

Choose an existing UR program variable whose value will be written to the local server.

Select the action to be performed when the write operation fails.

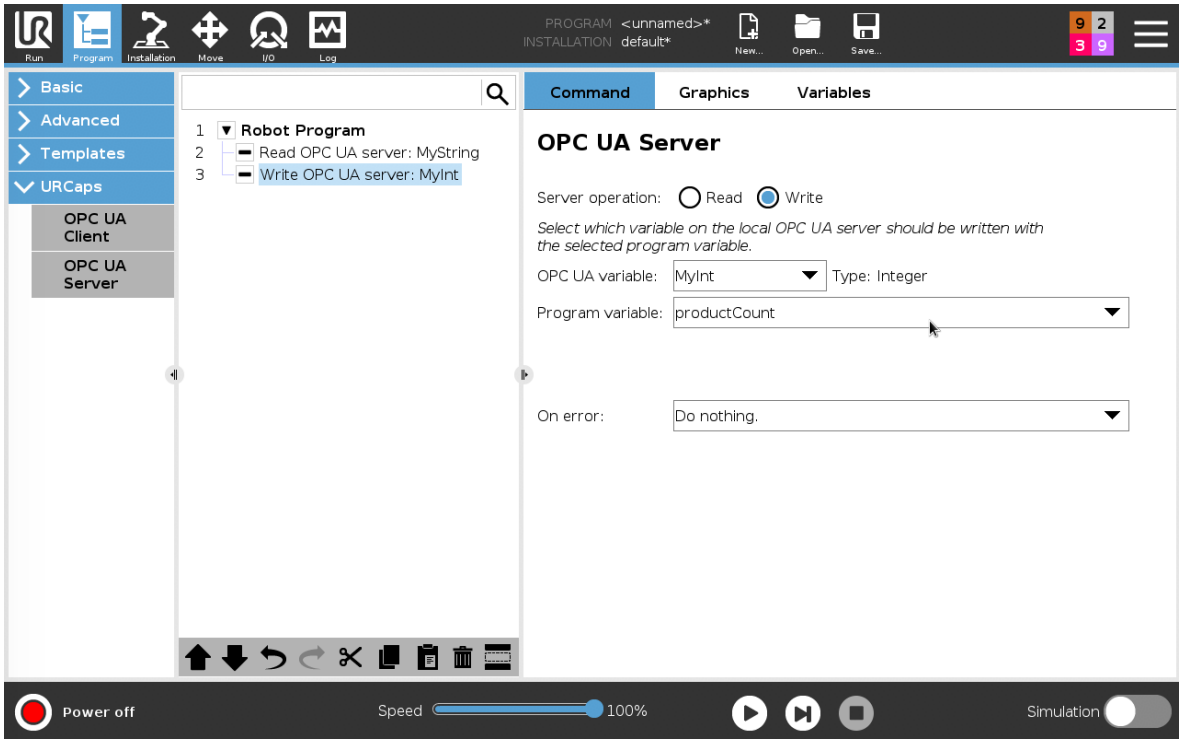

Figure 26 Server write operation

6.2 - Script functions

The following functions are available in the built-in Function Editor:

- opcua_client_connect
- opcua_client_get_id
- opcua_client_try_connect
- opcua_client_disconnect
- opcua_client_is_connected
- opcua_client_read_byid
- opcua_client_read_byname
- opcua_client_write_byid
- opcua_client_write_byname
- opcua_client_get_last_error
- opcua_server_read_byname
- opcua_server_write_byname
- opcua_server_get_last_error
- opcua_error_to_string

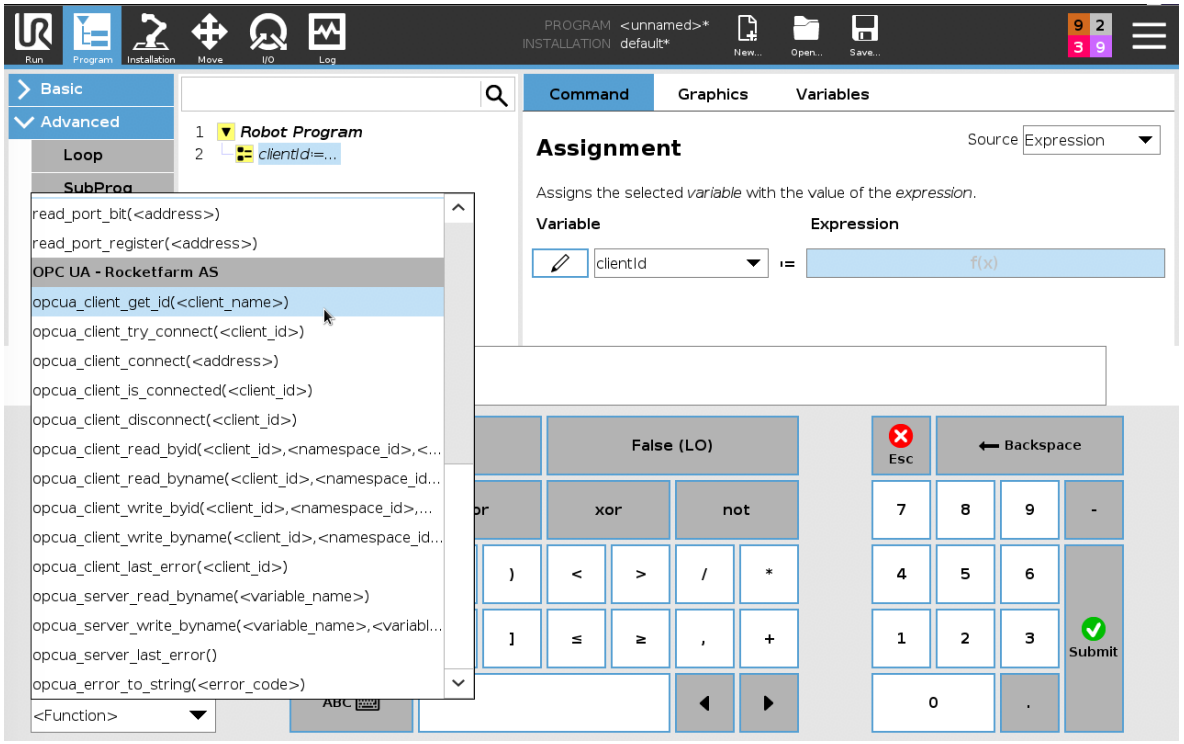

Figure 27 Using the built-in OPC UA functions in the Expression Editor

6.2.1 - opcua_client_connect

Use this function to connect directly to a remote OPC UA server. Security options and user authentication are not supported.

Syntax: client_id = opcua_client_connect("opc.tcp://address:port")

Return value: client id. On error, the return value is zero or negative.

6.2.2 - opcua_client_get_id

Use this function to obtain a client id of a client that is configured on the Installation page.

Syntax: client_id = opcua_client_get_id("clientName")

Return value: client id. On error, the return value is zero or negative.

Note: The "Client Connect" program node creates a variable with a name opcua_client_idX where X is the 0-based index of the connection in the Installation tab. This can be used as an alternative to the above function call.

6.2.3 - opcua_client_try_connect

Use this function to open a network connection to a remote server as a client. The client id is specified in the input parameter.

Syntax: result = opcua_client_try_connect(client_id)

Return value: True on success, False on error.

6.2.4 - opcua_client_disconnect

Use this function to close the network connection to a remote server. The client id is specified in the input parameter.

Syntax: result = opcua_client_disconnect(client_id)

Return value: True on success, False on error.

6.2.5 - opcua_client_is_connected

Syntax: result = opcua_client_is_connected(client_id)

Return value: True if the client is connected, otherwise False.

6.2.6 - opcua_client_read_byid

Use this function to read the value of a remote OPC UA server variable specified by a numerical node id.

Syntax: value = opcua_client_read_byid(client_id, namespace_id, id)

Return value: the value returned by the server.

6.2.7 - opcua_client_read_byname

Use this function to read the value of a remote OPC UA server variable specified by a string node name.

Syntax: value = opcua_client_read_byname(client_id, namespace_id, name)

Return value: the value returned by the server.

Note: Some OPC UA servers may use special characters in string identifiers, which could lead to syntax errors in the robot program. Use the '\' (backslash) character to enter the numerical value of such characters in a 4-digit hexadecimal form. Use \0022 to represent double quotes, and \005c for the backslash character itself.

6.2.8 - opcua_client_write_byid

Use this function to change the value of a remote OPC UA server variable specified by a numerical node id.

Syntax: result = opcua_client_write_byid(client_id, namespace_id, id, value)

Return value: True if successful, otherwise False.

6.2.9 - opcua_client_write_byname

Use this function to change the value of a remote OPC UA server variable specified by a string node name.

Syntax: result = opcua_client_write_byname(client_id, namespace_id, name, value)

Return value: True if successful, otherwise False.

Note: See note under opcua_client_read_byname.

6.2.10 - opcua_client_last_error

Use this function to obtain the last error code from the underlying open62541 module after a failed function call.

Syntax: error_code = opcua_client_last_error(client_id)

Return value: nonzero error code on error, or zero if the last function call was successful.

Note: More information about the error codes available here:
https://open62541.org/doc/current/statuscodes.html

6.2.11 - opcua_server_read_byname

Use this function to read the value of a local OPC UA server variable specified by a string node name.

Syntax: value = opcua_server_read_byname(name)

Return value: the value returned by the server.

6.2.12 - opcua_server_write_byname

Use this function to change the value of a local OPC UA server variable specified by a string node name.

Syntax: result = opcua_server_write_byname(name, value)

Return value: True if successful, otherwise False.

6.2.13 - opcua_server_last_error

Use this function to obtain the last error code from the underlying open62541 module after a failed function call.

Syntax: error_code = opcua_server_last_error()

Return value: nonzero error code on error, or zero if the last function call was successful.

Note: See note under opcua_client_last_error.

6.2.14 - opcua_error_to_string

Use this function to get detailed error text for a given numerical error code. The error text is returned in English.

Syntax: text = opcua_error_to_string(error_code)

Return value: The error text that corresponds to the specified error code.

6.3 - Low-level API (the OPC UA daemon)

Use the low-level functions of the OPC UA daemon to get full access to all functions of the OPC UA URCap.

6.3.1 - Technical data

Global variable for XMLRPC function calls	opcua_daemon
XMLRPC port number	40409, can be altered
Contributed id in *.installation	no.rocketfarm.urcap.opcua
Current version	2.0

6.3.2 - Diagnostic functions

Name	ping
Signature	bool ping()
Description	Test function. Always returns true.
Parameters	
Return value	true

6.3.3 - Client functions

Name	client_create
Signature	unsigned int client_create(string name, string address)
Description	Creates a client with the specified name and connection endpoint address
Parameters	string name: unique name of the client string address: endpoint address of the remote OPC UA server
Return value	unsigned int client_id: a positive number, a unique id of the newly created client; zero on error
Remarks	The client won't connect automatically until client_connect is performed.

Name	client_set_security_mode
Signature	bool client_set_security_mode(unsigned int client_id, unsigned int securityMode)
Description	Selects the message security mode to be used by the client.
Parameters	unsigned int client_id: unique identifier of the client unsigned int securityMode: message security mode to be used: ● 0: None ● 1: Sign ● 2: Sign and encrypt
Return value	true if successful, otherwise false
Remarks	The function has no effect when the client is already connected. Use this function before client_connect.

Name	client_set_security_policy
Signature	bool client_set_security_policy(unsigned int client_id, unsigned int securityPolicy)
Description	Selects the security policy to be used by the client.
Parameters	unsigned int client_id: unique identifier of the client unsigned int securityPolicy: security policy to be used: ● 0: None ● 1: Basic128Rsa15 ● 2: Basic256 ● 3: Basic256Sha256
Return value	true if successful, otherwise false
Remarks	The function has no effect when the client is already connected. Use this function before client_connect.

Name	client_set_username_and_password
Signature	bool client_set_username_and_password(unsigned int client_id, string username, string password)
Description	Sets the username and password to be used when connecting to the remote server.
Parameters	unsigned int client_id: unique identifier of the client string username: the login name to be used string password: encoded password, each character is represented as a 4-digit hexadecimal number
Return value	true if successful, otherwise false
Remarks	The function has no effect when the client is already connected. Use this function before client_connect.

Name	client_enable_authentication
Signature	bool client_enable_authentication(unsigned int client_id, bool enable)
Description	Changes the logon method to be used in the client connection. Switches between anonymous, and username + password authentication.
Parameters	unsigned int client_id: unique identifier of the client bool enable: true if username and password should be used, false if anonymous login should be used.
Return value	true if successful, otherwise false
Remarks	The function has no effect when the client is already connected. Use this function before client_connect.

Name	client_get_name
Signature	string client_get_name(unsigned int client_id)
Description	Returns the name of the specified client.
Parameters	unsigned int client_id: unique identifier of the client
Return value	string client_name: the name of the client identified by client_id

Name	client_get_id
Signature	unsigned int client_get_id(string client_name)
Description	Returns the unique identifier of the client with the specified name.
Parameters	string client_name: name of the client in question
Return value	unsigned int client_id: nonzero client identifier, or zero if not found

Name	client_connect
Signature	bool client_connect(unsigned int client_id)
Description	connects to a remote OPC UA server
Parameters	unsigned int client_id: the unique identifier of the client, must be obtained from the client_create function
Return value	TODO REFACTOR the return value is 0 or 1 not bool

Name	client_connect
Signature	unsigned int client_connect(string address)
Description	connects to a remote OPC UA server
Parameters	string address: the full address of the OPC UA server, e.g. opc.tcp://127.0.0.1:4840
Return value	unsigned int client_id: unique identifier of the client, zero if not successful
Remarks	simultaneously connecting to multiple servers are supported via client ids, see return value

Name	client_disconnect
Signature	bool client_disconnect(unsigned int client_id)
Description	disconnects from the given OPC UA server
Parameters	unsigned int client_id: identifier of the client to be disconnected
Return value	true if successful

Name	client_is_connected
Signature	bool client_is_connected(unsigned int client_id)
Description	returns true if the session is connected
Parameters	unsigned int client_id: unique identifier of the client
Return value	true if connected

Name	client_test
Signature	object[] client_test(string address)
Description	Internally used by automatic object discovery.
Parameters	string address: the address of the remote server to be tested
Return value	array of variable description objects, where each element contains ● namespace_id: the OPC UA namespace id of the variable ● is_numeric: true if the variable has numeric id, otherwise false ● variable_id: an integer or a string, depending on is_numeric ● description: the display name of the OPC UA variable

Name	client_read_variable_by_id
Signature	int/double/string client_read_variable_by_id(unsigned int client_id, int namespaceId, int variableId)
Description	reads the actual value of the given variable
Parameters	unsigned int client_id: unique identifier of the client

	int namespaceId: namespace identifier (see OPC UA) int variableId: variable identifier (see OPC UA)
Return value	returns the value of the specified variable (depending on type it can return int, double or string) or "NULL" if error occurred

Name	client_read_variable_by_name
Signature	int/double/string client_read_variable_by_name(unsigned int client_id, int namespaceId, string variableName)
Description	reads the actual value of the given variable
Parameters	unsigned int client_id: unique identifier of the client int namespaceId: namespace identifier (see OPC UA) string variableName: the OPC UA variable name
Return value	returns the value of the specified variable (depending on type it can return int, double or string) or "NULL" if error occurred
Remarks	See note about escaping special characters in string identifiers under 6.2.7 opcua_client_read_byname

Name	client_write_variable_by_id
Signature	bool client_write_variable_by_id(unsigned int client_id, int namespaceId, int variableId, int/double/string variableValue)
Description	writes the actual value of the given variable
Parameters	unsigned int client_id: unique identifier of the client int namespaceId: namespace identifier (see OPC UA) int variableId: the OPC UA variable id int/double/string variableValue: the new value to be written

Return value	true if successful, false otherwise

Name	client_write_variable_by_name
Signature	bool client_write_variable_by_name(unsigned int client_id, int namespaceId, string variableName, int/double/string variableValue)
Description	writes the actual value of the given variable
Parameters	unsigned int client_id: unique identifier of the client int namespaceId: namespace identifier (see OPC UA) string variableName: the OPC UA variable name int/double/string variableValue: the new value to be written
Return value	true if successful, false otherwise
Remarks	See note about escaping special characters in string identifiers under 6.2.7 opcua_client_read_byname

Name	client_get_last_error
Signature	int client_get_last_error(unsigned int client_id)
Description	Returns the last error of the underlying open62541 module, or zero if no error occurred.
Parameters	unsigned int client_id: unique identifier of the client
Return value	nonzero error code, or zero if the last function call succeeded.
Remarks	See note under opcua_client_last_error.

6.3.4 - Server functions

Name	server_start (deprecated)
Signature	bool server_start(int port=4840, bool fromFile=false, string fileNameOrConfig="opcua.ini")
Description	Loads the OPC UA server configuration from the given file or the given configuration, and starts the local OPC UA server on the given port.
Parameters	int port: the TCP port where the OPC UA server will listen bool fromFile: true if fileNameOrConfig contains a file name, false if it contains the configuration data itself string fileNameOrConfig: can contain the name of the ini-file, or the content itself (see the ini-file definition below)
Return value	true if successfully started.
Remarks	If the server is already running, it will be stopped and then started again.
Remarks	Configuration is a multi-line text, where each row describes one OPC UA variable. The row format is the following: name;type;defvalue where name: the name of the variable type: one of "int", "double", "string" defvalue: default value for the variable (on startup)

Name	server_init
Signature	bool server_init(int port=4840, bool fromFile=false, string fileNameOrConfig="opcua.ini")
Description	Loads the OPC UA server configuration from the given file or the given configuration, and creates the local OPC UA server on the given port, but *does not start the server*.

Parameters	int port: the TCP port where the OPC UA server will listen bool fromFile: true if fileNameOrConfig contains a file name, false if it contains the configuration data itself string fileNameOrConfig: can contain the name of the ini-file, or the content itself (see the ini-file definition below)
Return value	true if successfully created.
Remarks	If the server is already running, it will be stopped and then started again.
Remarks	Configuration is a multi-line text, where each row describes one OPC UA variable. The row format is the following: name;type;defvalue where name: the name of the variable type: one of "int", "double", "string" defvalue: default value for the variable (on startup)

Name	server_set_security_mode
Signature	bool server_set_security_mode(unsigned int securityMode, bool enabled)
Description	Enables or disables the specified message security mode of the server.
Parameters	unsigned int securityMode: the message security mode to be enabled or disabled: ● 0: None ● 1: Sign ● 2: Sign and encrypt bool enabled: true if the above specified mode is enabled, otherwise false
Return value	true on success, otherwise false
Remarks	The function has no effect when the local server is already running. Use this function before invoking server_run.

Name	server_set_security_policy
Signature	bool server_set_security_policy(unsigned int securityPolicy, bool enabled)
Description	Enables or disables the specified security policy of the server.
Parameters	unsigned int securityPolicy: the security policy to be enabled/disabled: ● 0: None ● 1: Basic128Rsa15 ● 2: Basic256 ● 3: Basic256Sha256 bool enabled: true if the above specified security policy is enabled, otherwise false
Return value	true on success, otherwise false
Remarks	The function has no effect when the local server is already running. Use this function before invoking server_run.

Name	server_set_authentication_policy
Signature	bool server_set_authentication_policy(unsigned int authenticationPolicy, bool enabled)
Description	Enables or disables the specified authentication policy of the server.
Parameters	unsigned int authenticationPolicy: the authentication policy to be enabled or disabled: ● 0: Anonymous ● 1: Username with password bool enabled: true if the above specified authentication policy is enabled, otherwise false
Return value	true on success, otherwise false
Remarks	The function has no effect when the local server is already running. Use this function before invoking server_run.

Name	server_add_user
Signature	bool server_add_user(string username, string password)
Description	Creates a user on the local server.
Parameters	string username: the login name to be created string password: encoded password, characters are represented in 4-digit hexadecimal numbers
Return value	true on success, otherwise false
Remarks	The function has no effect when the local server is already running. Use this function before invoking server_run.

Name	server_run
Signature	bool server_run()
Description	Starts the local OPC UA server that has been created and initialized by a preceeding call to server_init and other configuration functions.
Parameters	
Return value	Returns true if the server is started, otherwise false.

Name	server_is_running
Signature	bool server_is_running()
Description	Returns the running status of the local OPC UA server.
Parameters	

Return value	Returns true if the server is running, otherwise false.

Name	server_stop
Signature	bool server_stop()
Description	Stops the running server and deletes all underlying data structures.
Parameters	
Return value	true

Name	server_read_variable_by_name
Signature	int/double/string server_read_variable_by_name(string attrName)
Description	reads the given variable of the local OPC UA server
Parameters	string attrName: name of the variable (see ini file under server_start)
Return value	int value: 32-bit integer value double value: double-precision floating point value string value: text

Name	server_write_variable_by_name
Signature	bool server_write_variable_by_name(string attrName, int/double/string value)
Description	writes the given variable
Parameters	string attrName: name of the variable (see ini file under server_start) int value: 32-bit integer value

	double value: double-precision floating point value string value: text
Return value	true if successful, otherwise false

Name	server_get_last_error
Signature	int server_get_last_error()
Description	Returns the last error of the underlying open62541 module, or zero if no error occurred.
Parameters	
Return value	nonzero error code, or zero if the last function call succeeded.
Remarks	See note under opcua_client_last_error.

6.3.5 - Utility functions

Name	error_code_to_string
Signature	string error_code_to_string(int error_code)
Description	Returns the English error text that corresponds to the specified OPC UA error code.
Parameters	int error_code: the OPC UA error code that needs to be described.
Return value	string text: the description of the error in English

Name	get_current_timestamp
Signature	int get_current_timestamp()
Description	Returns the current timestamp - seconds since 1970.01.01. 00:00:00 GMT - as an integer.
Parameters	
Return value	int result: seconds since 1970.01.01. 00:00:00 GMT.

Name	get_current_datetime
Signature	int[] get_current_datetime()
Description	Returns the current local date and time as an array of integers.
Parameters	
Return value	The returned array elements are: 0: year

	1: month (1-12) 2: day of the month (1-31) 3: hour (0-23) 4: minutes (0-59) 5: seconds (0-59) 6: day of the week (1: monday,.. 7: sunday) 7: day of the year (1-366)

Name	get_current_datetime_utc
Signature	int[] get_current_datetime_utc
Description	Returns the current UTC date and time as an array of integers.
Parameters	
Return value	The same structure as get_current_datetime, but in UTC instead of the local time zone.

Name	concat
Signature	string concat(bool/int/double/string value1, value2, ...)
Description	Concatenates all input parameters into a string.
Parameters	bool/int/double/string valueN: a value that can be boolean, integer, double, or string type. The number of parameters is not limited.
Return value	The input parameters are concatenated into a string.

Name	substring

Signature	string substring(string source, int startPos=0, int length=-1)
Description	Returns a substring of the source string. This is a wrapper for the string::substr() C++ function.
Parameters	string source: the source string int startPos: optional, zero-based position in the source string int length: optional, length of the substring, or "until the end of the string" when not given.
Return value	A substring of the given string. For further details see http://www.cplusplus.com/reference/string/string/substr/

Name	strfind
Signature	int strfind(string source, string sequence, int startPos=0)
Description	Finds a substring in a string. Returns the index of the first character. This is a wrapper for the string::find() C++ function.
Parameters	string source: the source string string sequence: the string to be found in the source string int startPos: the zero-based position from where the search should begin
Return value	Zero-based index of the first occurrence of the sequence in the source string. If the sequence cannot be found, the return value is -1.

Currently supported variable types: bool, string, integer, double.

Arrays are not supported for now, but are scheduled to be implemented in a release in the near future.

7 - OPC Robotics

The URCap conforms to the OPC VDMA 40010 Robotics Companion Specification v1.0. Objects and variables of the OPC Robotics Information Model are automatically published when the server is enabled. No additional configuration is required.

Currently the following objects and variables are available.

7.1 - The root object

The root object is a MotionDeviceSystemType object that is called "Robot" and is located under the "Objects" node in the server. The Robot object contains the following 3 folders:

- MotionDevices
- Controllers
- SafetyStates

Each folder will be described in the next sections.

7.2 - MotionDevices

This folder contains one MotionDeviceType object that is called "RobotArm". It corresponds to the physical robot arm connected to the controller that runs the URCap.

Here you can find basic information about the robot arm:

- Manufacturer: "Universal Robots"
- Model: value depends on the actual robot model as derived from the serial number e.g. "UR10e 10Kg"
- MotionDeviceCategory: 1 (ARTICULATED_ROBOT)
- ProductCode: value depends on the actual robot model as derived from the serial number e.g. "UR10e"
- SerialNumber: the serial number of the robot e.g. "20185099999"
- ParameterSet:
    - SpeedOverride: position of the speed slider in Polyscope as a percentage

7.2.1 - Axes

This folder contains 6 AxisType objects that correspond to the axes of the robot arm:

- Base
- Shoulder
- Elbow
- Wrist1
- Wrist2
- Wrist3

Each entry contains the following variables:

- MotionProfile: 1 (ROTARY)
- ParameterSet:
    - ActualPosition: position of the corresponding joint in radians
    - ActualSpeed: speed of the corresponding joint in rad/sec

7.2.2 - PowerTrains

This folder contains 6 MotorType objects that correspond to the motors in the robot arm:

- Base
- Shoulder
- Elbow
- Wrist1
- Wrist2
- Wrist3

Each entry contains the following parameters:

- Manufacturer: "Universal Robots"
- Model: text "Size 0" to "Size 5" depending on the motor size
- ProductCode: part number as specified in the Universal Robots Service Manual
- SerialNumber: unique identifier generated by appending a 4-digit counter at the end of the robot serial number, e.g. "201850999990002"
- ParameterSet:
    - MotorTemperature: temperature of the corresponding motor in Cesius

7.3 - Controllers

This folder contains one ControllerType object that is called "ControlBox". It corresponds to the controller that runs the URCap.

Here you can find basic information about the robot arm:

- Manufacturer: "Universal Robots"
- Model: value depends on the actual controller as derived from the serial number e.g. "Control Box for UR5e/UR10e/UR16e"
- ProductCode: part number as specified in the Universal Robots Service Manual e.g. "102400"
- SerialNumber: the serial number of the controller e.g. "20185099999"
- CurrentUser:
    - Level: a string returned by the Dashboard Server

7.3.1 - Software

This folder contains 3 SoftwareType objects that correspond to the software components of the controller:

- Backend
- Frontend
- OperatingSystem

Each entry contains the following variables:

- Manufacturer
- Model
- SoftwareRevision

The data is filled automatically by executing various low-level system commands.

7.3.2 - TaskControls

This folder contains one TaskControlType object that is called "RobotProgram" with the following variables:

- ComponentName: a user-writable string as required by the standard, currently empty
- ParameterSet:
    - TaskProgramLoaded: true if a program is loaded in Polyscope, otherwise false
    - TaskProgramName: full path of the program name currently loaded in Polyscope, e.g. "/programs/palletizer.urp"

7.4 - SafetyStates

This folder contains one SafetyStateType object that is called "SafetyState" with the following variables:

- ParameterSet:
    - EmergencyStop: true if the robot is emergency stopped, otherwise false
    - ProtectiveStop: true if the robot is protective stopped, otherwise false
    - OperationalMode: 3 while the robot program is running, 2 when not running, 0 when the robot state is unknown (e.g. during system startup, etc.)