IRA\_UR\_SocketCtrl\_Prog   
C# UR HW Driver

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Doc. Revision | Date | Name | SW Revision | Modification Reason |
| A | 2023-09-04 | William Weiske | 1.0.0.0 | Document creation |
| B | 2023-10-30 | William Weiske | 1.1.0.0 | Added Features, Bugfixes, GUI Update |
|  |  |  |  |  |

Table of contents

[Scope 3](#_Toc144709063)

[Target Audience 3](#_Toc144709064)

[Requirements 3](#_Toc144709065)

[Debug GUI and Launcher 3](#_Toc144709066)

[Launch program 3](#_Toc144709067)

[Debug GUI 4](#_Toc144709068)

[Connection 4](#_Toc144709069)

[Robot Info / Command display 5](#_Toc144709070)

[Movement and Actions 6](#_Toc144709071)

[Settings 10](#_Toc144709072)

[IRA\_UR\_SocketCtrl\_Prog 11](#_Toc144709073)

[Setup functions and Debug GUI 11](#_Toc144709074)

[Properties 11](#_Toc144709075)

[Basic 11](#_Toc144709076)

[Robot 12](#_Toc144709077)

[Joints and Angles 13](#_Toc144709078)

[IOs 13](#_Toc144709079)

[Tool Data 14](#_Toc144709080)

[Decoded raw data 14](#_Toc144709081)

[Settings 15](#_Toc144709082)

[Output Functions 16](#_Toc144709083)

[Digital Outputs 16](#_Toc144709084)

[Analog Outputs 16](#_Toc144709085)

[Move Functions 17](#_Toc144709086)

[MoveJ 17](#_Toc144709087)

[MoveL 19](#_Toc144709088)

[Stop Functions 21](#_Toc144709089)

[Special Functions 21](#_Toc144709090)

[Move Wait 21](#_Toc144709091)

[Freedrive 22](#_Toc144709092)

[Sending Commands(Expert use only) 22](#_Toc144709093)

[SendCommand (Default to Port 30002) 22](#_Toc144709094)

[SendCommandInt (Default to Port 30020) 23](#_Toc144709095)

[Advanced Mode (Expert use only) 23](#_Toc144709096)

[Functions 23](#_Toc144709097)

[Properties 23](#_Toc144709098)

# Scope

This document provides details for how the use IRA UR Socket Control Driver. IRA\_UR\_SocketCtrl\_Prog is written in C# (.NET) which can be used in a wide range of different Automation Tools.

An additional Debug GUI can be used to test implemented features. It can also be used stand alone to move the robotic arm remotely without automation.

# Target Audience

NPI (New Product Introduction Team), IRA (Internal Robot and Automation Team)

# Requirements

URxe-Series Robot with firmware version V. 5.12 or later.

Ethernet connection to Robot Controller

Window PC (Win10 or above)

# Debug GUI and Launcher

## Launch program

To use the Debug GUI you first need to start it’s launcher “IRA\_UR\_SocketCtrl\_Prog\_Launcher.exe”.

Ein Bild, das Text, Screenshot, Zahl, Display enthält.

Automatisch generierte Beschreibung

Figure 1 – IRA\_UR\_SocketCtrl\_Prog\_Launcher.exe

The Launcher has two options:

* Integrated use (e.g. TestStand)
  + This sets the driver up as it would be in an automated solution.
    - Create ObjectReference   
      SocketCtrl(string hostIp = "192.168.1.251", int port = 30002, int portInt = 30020, int cyclePause = 100, int timeout = 1000)
    - Start()
    - [Actions to do]
    - Stop()
  + ShowDbgGUI()
    - This might require some more implementation work as the sequence needs to wait that GUI was closed.
    - Remarks:   
      There is a high chance that it will not work properly in the beginning as this function was tested with C# GUI only.   
      Other applications, sequencers might require an improvement.
    - CheckDbgGUIClosed()
      * Use this to check if GUI was closed.
* StandAlone
  + Use this option to test your environment

## Debug GUI

The Debug GUI has several sections:

* Connection
* Robot Info / Command display
* Movement and Actions
* Settings

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

### Connection

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Connection section includes Robot IP Address, generic port, interpreter port and cycle pause time (time to wait before reading again from socket).

“Start()” and “Stop()” are opening and closing connection to robot.

* These options are unavailable in integrated use as this is controlled by the calling program.

“Runtime” represents the robot’s uptime since last boot.

### Robot Info / Command display

Most required robot stats are displayed in the info text section. All data can be gathered from raw data objects. See driver documentation.

To select data from text it’s easier to “hold” it by unchecking “Update”.

If “Show last command” is checked then it will show the last command sent to the robot.

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

### Movement and Actions

This section includes all current available functions for movement. To use it properly please start from the top left corner.

Ein Bild, das Text, Screenshot, Diagramm, Display enthält.

Automatisch generierte Beschreibung

#### Movement Mode

Movement Mode specifies how the robot should make it’s path. Please see robot documentation for more details.

* “Joints” will set angle of each joint.
* “Pose Axis Angle” will set position in coordinate system and rotation vector.
* “Pose Roll-Pitch-Yaw” selection will clear the tools rotation to “0, 0, 0” which means that the tool flange will face up vertical from base as there is actually no solution to convert it back from actual pose to RPY. Besides that it allows to easily and understandable move the tool in 3D space.

Movement methods:

* movej – linear in joint space
* movel – linear in tool space
* movec – circular in tool space 🡪 this requires more parameter, input in [special] box
* speedl – accelerate in tool-space 🡪 this requires more parameter, input in [special] box
* MoveContact – Moves the robot until contact is made 🡪 this requires more parameter, input in [special] box

Freedrive comes in two variants:

* Checkbox – Robot will stay in Freedrive mode as long as it’s checked, allowing the operator to have two hands free.
* Button – Like the classic way – hold it to move it.

Remarks:

* Robot might start to move if target payload and/or TCP offset are not set properly!
* Freedrive requires Advanced Mode which can interfere with a running advanced sequence. Use with care!

#### Step Settings

These parameters define how a step or move will behave.

“Pos” and “Angle” are corresponding with Move section for “+” and “-” features. They are only part of Debug GUI.

“a”, “v”, “r”, “aRot” are parameters parsed to the robot. Please see robot documentation for more details. “r” is for blending movements with a specified radius and can be used only when in Advanced Mode as it requires a continuous movement over n+1 waypoints. The last waypoint must be without blend radius or robot will stop immediately. Please see robot documentation for more details.

“Angle in Rad” will use rotation in radiant if checked, else will use degree. This also effects all robot stats in info text section. Angles will be converted anyway to radiant before parsed to robot.

“Wait complete” is designed to keep the function called as long as the robot is active and return when the execution ended. Remark: Some feature can cause a deadlock if “Wait complete” is active.

#### Move While Condition

Enable this feature to execute movements in dependency of input signals. A movement will run until it reaches its target if the condition is true. If the condition turns to false then movement will be stopped immediately.

Select input source “tool-di, std-di, tool-ai, std-ai” (di 🡪 digital input, ai 🡪 analog input). Analog or digital value will always be parsed but will be selected in driver when building the condition.

Number specifies the bit and valid range is 0…7 (not all sources have that many channels):

* tool-di [0…1]
* std-di [0…7]
* tool-ai [0…1]
* std-ai [0…1]

Comparison defines how to compare (not all data types make sense with all operators):

* == - equal
* != - not equal
* > - greater than
* < - less than
* >= - greater equal than
* <= - less equal than

Analog value in range [0.0 … 1.0] for analog input. Slider and textbox are synchronized. Hit enter in textbox after new value is typed in.

Boolean value [true, false] for digital input. Check or uncheck.

This functionality is not available for MoveContact.

#### Move

The Move section is the area to actually move the robotic arm to a coordinate and rotation. The usage of these fields depends on the selection made in the “Movement Mode” and “Step Settings” sections. Definitions are as follows:

* Joints [base, shoulder, elbow, wrist1, wrist2, wrist3]
* Pose Axis Angle [x, y, z, rx, ry, rz]
* Pose Roll-Pitch-Yaw [x, y, z, roll, pitch, jaw]
* SpeedL [x, y, z, rx, ry, rz] – this uses speed instead of positions
* MoveContact [x, y, z, rx, ry, rz] – this uses speed instead of positions

“Copy Robot” copies the current position depending on the settings to the textboxes. If “Pose RPY” is selected it will clear the tools rotation vector to “0, 0, 0” which means that the tool flange will face vertical from base as there is actually no solution to convert it back from actual pose to RPY.

“Stop” will stop the robot immediately with deceleration “a”.

“Move Robot” parses the current coordinates to the robot.

“+” and “-” will increment / decrement the current coordinates by “Pos” and “Angle” and selected movement.

Every button hit will be executed if “Auto Move” is checked. “Auto Move” will also be unchecked by default when either “movec”, “speedl” or “MoveContact” are selected.

#### [special]

This section is used dynamically if more parameters are required than a regular movement needs. The following function use it as following:

* movec – “Path via point”   
  The circular movement requires an additional coordinate along the circular path to the target point set in “Move”. This position is called the “via” point. It can be either relative to the starting point (if “relative” is checked) or as global coordinate in tool space. By selecting “movec” or clicking “Copy Robot” the current coordinates from the robot’s position are copied into the textboxes.
  + The via point only accepts x,y,z coordinate.
  + Mode defines how the rotation will be used.
    - 0 := Unconstrained mode: interpolates from current pose to target pose
    - 1 := Fixed mode: Keep orientation constant relative to the tangent of the circular arc from starting point
* speedl – “Time until function returns”   
  “speedl” accelerates the TCP in direction and speed depending on the values set in “Move”. The function returns by default when the target speed is met. (time = 0)
  + If a time is set in seconds then the function will return even if the target speed is not met. When time is longer than the acceleration the movement will be constant until time has passed.
  + “aRot” is an optional parameter. It is used for angular acceleration if set.
* MoveContact – “Direction and max length of detection path”  
  “MoveContact” is a function to detect an obstacle without causing a collision stop in Controller. This function is very delicate in some direction depending on the pose.
  + As the core part of this function uses the “speedl” behavior it expects the parameters set in “Move” as speed and direction.
  + The “[special]” section then defines in which direction a possible obstacle is expected. All ”0.0000” will allow any contact to stop the movement. This includes a maximum distance from starting point to stop if nothing was detected. The calculation for stop is based on the 3D distance independent from the direction to keep calculations at a minimum. This means that a rotation with no x,y,z-move will not properly work.

### IOs

Ein Bild, das Text, Screenshot, Display, Zahl enthält.

Automatisch generierte Beschreibung

#### Output

Output section holds all available outputs for direct control.

Analog values in range [0.0 … 1.0] per output. Slider and textbox are synchronized. Hit enter in textbox after new value is typed in.

Boolean values [true, false] for digital outputs. Check or uncheck.

“Read Back” load the current state from robot stats.

### Read Data

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

#### TCP Force

The actual TCP Force can be read out from controller.

Remarks:

* The values are not precise and can differ from cycle to cycle depending on robot pose.

### Advanced + More

This section contains functions that are more than only movements.

Ein Bild, das Text, Screenshot, Display, Schrift enthält.

Automatisch generierte Beschreibung

#### Advanced Mode (Expert use)

Advanced Mode allows to create a linear sequence of actions the robot can execute. Not for if-else logic but instead for complex movement paths with blend radius. Move While Conditions are allowed but make most sense as last step. Setting outputs in sequence is also possible.

“Enable” sets the robot into interpreter mode which allows recording of actions to the list.

Actions can be added to the list like usual robot movement but instead of direct execution they will be recorded for later. To display the current list just hover over the listbox below the buttons.

“Execute” will format the command queue and send it as one program to the interpreter port. All steps are removed after execution. (direct usage of driver has an option to keep the list)

“Clear Queue” clears the list.

“Exit” stops the interpreter and exits the Advanced Mode.

#### Send Script File

This button will open a file dialog to select a UR Script File which will be send to robot and executed in controller. The script file must be UR Script compliant or it will be ignored by the controller. To keep track of the execution it can be combined with “Move -> Step Settings -> Wait Complete”.

### Settings

Settings allow to change robot parameters.

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

#### TCP Offset

It is recommended that the robot knows where the Tool Center Point is to reach the right position. If the offset is wrong then movements might fail like rotation center incorrect, hitting other things as “end” of tool is longer than set.

Structure is represented as [x, y, z, rx, ry, rz]. Please see robot documentation for more details.

“Angle as Rad” is independent from the Movement section.

“Set TCP Offset” parses the coordinates to robot.

#### Target Payload

Target Payload describes the “structure” and mass of an object hold by the robot. If settings are wrong then robot might behave wrong, will stop operation as force vector might exceed safety settings or enabled Freedrive will cause the robot to move.

“Mass” defines the mass of the object. It is the most important parameter.

“Center of Gravity” defines where the mass is located from the tool flange.

“Inertia” is an optional parameter which might be used for special use cases. Setting it to “0” will ignore it.

“Set Target Payload” parses the settings to robot.

# IRA\_UR\_SocketCtrl\_Prog

The core part of this driver is the SocketCtrl class which contains all major functionality.

Some of the code examples are shorted and not actual C# code but represent the most important information inherited.

## Setup functions and Debug GUI

Functions required to setup SocketCtrl

Create SocketCtrl object

< "hostIp">Host IP of Robot</ >

<"port">Generic port</ >

<"portInt">Interpreter port</ >

<"cyclePause">Cycle pause to wait before next read from socket</ >

<"timeout">Timeout for Robot response for connection</ >

public SocketCtrl(string hostIp = "192.168.1.251", int port = 30002, int portInt = 30020, int cyclePause = 100, int timeout = 1000)

Start Client Socket and connect

<returns>Result object</returns>

public Result Start()

Stop Client Socket

public void Stop()

Show Debug GUI

public void ShowDbgGUI()

Check if Debug GUI was closed or not opened

<returns>True := When closed</returns>

public bool CheckDbgGUIClosed()

## Properties

Access Properties to get data from current run state.

### Basic

Connection data and timing.

Read IP addresse back

public string HostIp { get; }

Read Port back

public int Port { get; }

Read Port of interpreter mode back

public int PortInt { get; }

Read Cycle Pause

public int CyclePause { get; }

Read actual Cycle Time

public int CycleTime { get; }

Read Connection Max Timeout back

public int Timeout { get; }

True := When robot socket read

public bool IsRunning { get { return running; } }

### Robot

Information about robot.

Incrementing Timestamp since Robot was booted

public ulong Timestamp { get { return RMD0.timestamp; } }

Physical Robot connected

public bool IsRealRobotConnected { get { return RMD0.isRealRobotConnected; } }

Physical Controller present

public bool IsRealRobotEnabled { get { return RMD0.isRealRobotEnabled; } }

Board power on

public bool IsRobotPowerOn { get { return RMD0.isRobotPowerOn; } }

EM Stop is active

public bool IsEmergencyStopped { get { return RMD0.isEmergencyStopped; } }

Softstop is active

public bool IsProtectiveStopped { get { return RMD0.isProtectiveStopped; } }

Program or action is in execution

public bool IsProgramRunning { get { return RMD0.isProgramRunning; } }

Program is halted in execution

public bool IsProgramPaused { get { return RMD0.isProgramPaused; } }

Controller Board Temperature

public float MasterBoardTemperature { get { return MBD3.masterBoardTemperature; } }

Controller Board Voltage

public float RobotVoltage48V { get { return MBD3.robotVoltage48V; } }

Controller Board Input Current

public float RobotCurrent { get { return MBD3.robotCurrent; } }

### Joints and Angles

Get pose / joints of robotic arm.

Get Actual Joint Angles as array size 6 in RAD

public double[] ActualJointAnglesRad

{ get { return new double[] { JD1.joints[0…5].q\_actual }; } }

Get Actual Joint Angles as array size 6 in DEG

public double[] ActualJointAnglesDeg

{ get { return new double[] { JD1.joints[0…5].q\_q\_actual\_deg}; } }

Get Actual Pose from Cartesian space as array size 6 with rotation vector in RAD

public double[] ActualPoseCartesianRad

{ get { return new double[] { CI4.X, CI4.Y, CI4.Z, CI4.Rx, CI4.Ry, CI4.Rz }; } }

Get Actual Pose from Cartesian space as array size 6 with rotation vector in DEG

public double[] ActualPoseCartesianDeg

{ get { return new double[] { CI4.X, CI4.Y, CI4.Z, CI4.RxDeg, CI4.RyDeg, CI4.RzDeg }; } }

Get TCP Offset from Cartesian space as array size 6 with rotation vector

public double[] TCPOffsetCartesian

{ get { return new double[] { CI4.TCPOffsetX, CI4.TCPOffsetY, CI4.TCPOffsetZ, CI4.TCPOffsetRx, CI4.TCPOffsetRy, CI4.TCPOffsetRz }; } }

Get actual TCP Force calibration data array size 6 [Fx, Fy, Fz, Frx, Fry, Frz]

public double[] ActualTCPForce

{ get { return new double[] { CalD9.Fx, CalD9.Fy, CalD9.Fz, CalD9.Frx, CalD9.Fry, CalD9.Frz }; } }

### IOs

Read IO states

Get Digital Output State

public bool[] DigitalOutputState

{ get { return GetBitsFromInt(MBD3.digitalOutputBits, 0); } }

Get Digital Input State

public bool[] DigitalInputState

{ get { return GetBitsFromInt(MBD3.digitalInputBits, 0); } }

Get Configurable Output State

public bool[] ConfigOutputState

{ get { return GetBitsFromInt(MBD3.digitalOutputBits, 1); } }

Get Configurable Input State

public bool[] ConfigInputState

{ get { return GetBitsFromInt(MBD3.digitalInputBits, 1); } }

Get Reserved Output State

public bool[] ReservedOutputState

{ get { return GetBitsFromInt(MBD3.digitalOutputBits, 3); } }

Get Reserved Input State

public bool[] ReservedInputState

{ get { return GetBitsFromInt(MBD3.digitalInputBits, 3); } }

Get Analog Output values

public double[] AnalogOutput

{ get { return new double[] { MBD3.analogOutput0, MBD3.analogOutput1 }; } }

Get Analog Input values

public double[] AnalogInput

{ get { return new double[] { MBD3.analogInput0, MBD3.analogInput1 }; } }

### Tool Data

Tool related data.

Get Tool Digital Output State

public bool[] ToolOutputState

{ get { return GetBitsFromInt(MBD3.digitalOutputBits, 2); } }

Get Tool Digital Input State

public bool[] ToolInputState

{ get { return GetBitsFromInt(MBD3.digitalInputBits, 2); } }

Get Tool Analog Input values

public double[] ToolAnalogInput

{ get { return new double[] { TD2.analogInput0, TD2.analogInput1 }; } }

Get Tool Temperature

public float ToolTemperature { get { return TD2.toolTemperature; } }

Get Tool Voltage

public float ToolVoltage48V { get { return TD2.toolVoltage48V; } }

Get Tool Current

public float ToolCurrent { get { return TD2.toolCurrent; } }

### Decoded raw data

All available raw data gets converted to objects holding them in a representable format. Some date might need to be further processed after accessing these objects.

Basic data for current robot state

public RDP.RobotModeData\_0 RobotModeData\_0 { get { return RMD0; } }

Data of robotic arm position and state

public RDP.JoinData\_1 JoinData\_1 { get { return JD1; } }

Data of robot tool

public RDP.ToolData\_2 ToolData\_2 { get { return TD2; } }

Controller board data, power and IOs

public RDP.MasterBoardData\_3 MasterBoardData\_3 { get { return MBD3; } }

Info of robotic arm position and TCP offset

public RDP.CartesianInfo\_4 CartesianInfo\_4 { get { return CI4; } }

Robot arm joint parameters

public RDP.KinematicsInfo\_5 KinematicsInfo\_5 { get { return KI5; } }

Robot arm joint configuration

public RDP.ConfigurationData\_6 ConfigurationData\_6 { get { return ConD6; } }

Force mode configuration data

public RDP.ForceModeData\_7 ForceModeData\_7 { get { return FMD7; } }

Addtional robot information and freedrive

public RDP.AdditionalInfo\_8 AdditionalInfo\_8 { get { return AI8; } }

Calibration data of robot arm

public RDP.CalibrationData\_9 CalibrationData\_9 { get { return CalD9; } }

Safety data (not decoded)

public RDP.SafetyData\_10 SafetyData\_10 { get { return SD10; } }

Tool communication parameters (serial data)

public RDP.ToolCommunicationInfo\_11 ToolCommunicationInfo\_11 { get { return TCI11; } }

Tool mode settings (DIOs)

public RDP.ToolModeInfo\_12 ToolModeInfo\_12 { get { return TMI12; } }

Singularity info (never seen that initialized)

public RDP.SingularityInfo\_13 SingularityInfo\_13 { get { return SI13; } }

## Settings

Settings are meant to adjust robot configurations. The robot’s operation might be affected if settings are wrong and can also lead to harming the machine itself and also people. Changing settings directly effects the operation without any confirmation! Please see robot documentation for more details.

Set TCP Offset

<"mass">Mass in kg</ >

<"cog">Center of Gravity in meter [CoGx, CoGy, CoGz]</ >

<"inertia">Optional: Inertia matrix in kg\*m² [Ixx, Iyy, Izz, Ixy, Ixz, Iyz]</ >

<returns>Result object</returns>

public Result SetTcpOffset(double[] pose, bool angleInRAD)

Set Target Payload

<"mass">Mass in kg</ >

<"cog">Center of Gravity in meter [CoGx, CoGy, CoGz]</ >

<"inertia">Optional: Inertia matrix in kg\*m² [Ixx, Iyy, Izz, Ixy, Ixz, Iyz]</ >

<returns>Result object</returns>

public Result SetTargetPayload(double mass, double[] cog, double[] inertia = null)

## Output Functions

Outputs can be controlled directly. There are different functions for separate categories (Tool, Standard, Configurable)

Outputs can be either digital or analog.

### Digital Outputs

Digital outputs can be either true (1) or false (0).

Set standard digital output signal level

<"bit">The number (id) of the output, integer: [0:7]</ >

<"state">The signal level. (boolean)</ >

<returns>Result object</returns>

public Result SetStandardDigitalOut(int bit, bool state)

Set tool digital output signal level

<"bit">The number (id) of the output, integer: [0:1]</ >

<"state">The signal level. (boolean)</ >

<returns>Result object</returns>

public Result SetToolDigitalOut(int bit, bool state)

Set configurable digital output signal level

<"bit">The number (id) of the output, integer: [0:7]</ >

<"state">The signal level. (boolean)</ >

<returns>Result object</returns>

public Result SetConfigurableDigitalOut(int bit, bool state)

### Analog Outputs

Analog outputs have a range from 0…1 meaning 0…100% independent from configuration.

Set standard analog output signal level

<param name="port">The number (id) of the output, integer: [0:1]</ >

<param name="value">

The relative signal level [0...1] (float)

double = 1.0, that corresponds to 10V (or 20mA depending on domain setting) on the output port

</ >

<returns>Result object</returns>

public Result SetStandardAnalogOut(int port, double value)

## Move Functions

Move Functions are the core part of the robot actions. There are several types of how to control it. Some allow to stop operation when a certain input leaves a required condition.

The functions below are over loaded with different parameters to allow easy access of feature sets.

Please see robot documentation for more details.

### MoveJ

Move to position in linear in joint-space will let the robot move with most less movement required on each joint. The path will look rounded from the outside.

Move to position linear in joint-space

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz]

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s

<"v">Velocity in rad/s

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<returns>Result object</returns>

public Result MoveJ(double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0)

Move to position linear in joint-space from Roll-Pitch-Jaw

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s</ >

<"v">Velocity in rad/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<returns>Result object</returns>

public Result MoveJ(double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0)

Move to position linear in joint-space, move while condition is true

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz] <para />

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s

<"v">Velocity in rad/s

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveJ(double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

Move to position linear in joint-space from Roll-Pitch-Jaw, move while condition is true

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s</ >

<"v">Velocity in rad/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveJ(double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

### MoveL

Move to position in linear in tool-space will let the robot move linear from operator view. The movement required on each joint might be higher.

Move to position linear in tool-space

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz]

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<returns>Result object</returns>

public Result MoveL(double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0)

Move to position linear in tool-space from Roll-Pitch-Jaw

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<returns>Result object</returns>

public Result MoveL(double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0)

Move to position linear in tool-space, move while condition is true

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz]

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s</ >

<"v">Velocity in rad/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveL(double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

Move to position linear in tool-space from Roll-Pitch-Jaw, move while condition is true

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1 <para />

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveL(double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

### MoveC

Move circular in linear in tool-space will let the robot move linear from operator view. The movement required on each joint might be higher. This set of functions requires additional parameters for the “via”-point.

Move circular in tool-space

<"poseVia">Path Point, only position is used [m] => [x, y, z, 0, 0, 0]</ >

<"relativeVia">True := Path Point is relative to start point</ >

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz]

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"mode">

Mode for orientation

0 := Unconstrained mode. Interpolate orientation from current pose to target pose (pose\_to)

1 := Fixed mode. Keep orientation constant relative to the tangent of the circular arc (starting from current pose)

</ >

<returns>Result object</returns>

public Result MoveC(double[] poseVia, bool relativeVia, double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, int mode = 1)

Move circular in tool-space from Roll-Pitch-Jaw

<"poseVia">Path Point, only position is used [m] => [x, y, z, 0, 0, 0]</ >

<"relativeVia">True := Path Point is relative to start point</ >

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"mode">

Mode for orientation

0 := Unconstrained mode. Interpolate orientation from current pose to target pose (pose\_to)

1 := Fixed mode. Keep orientation constant relative to the tangent of the circular arc (starting from current pose)

</ >

<returns>Result object</returns>

public Result MoveC(double[] poseVia, bool relativeVia, double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, int mode = 1)

Move circular in tool-space, move while condition is true

<"poseVia">Path Point, only position is used [m] => [x, y, z, 0, 0, 0]</ >

<"relativeVia">True := Path Point is relative to start point</ >

<"target">

Target TCP as pose in meter, radiant [x, y, z, rx, ry, rz]

Target as joint positions [base, shoulder, elbow, wrist1, wrist2, wrist3]

</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"asPose">

True := Target is defined as pose

False := Use as joint positions (angles)

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in rad/s/s</ >

<"v">Velocity in rad/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"mode">

Mode for orientation

0 := Unconstrained mode. Interpolate orientation from current pose to target pose (pose\_to)

1 := Fixed mode. Keep orientation constant relative to the tangent of the circular arc (starting from current pose)

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveC(double[] poseVia, bool relativeVia, double[] target, bool waitComplete = false, bool asPose = true, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, int mode = 1, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

Move circular in tool-space from Roll-Pitch-Jaw, move while condition is true

<"poseVia">Path Point, only position is used [m] => [x, y, z, 0, 0, 0]</ >

<"relativeVia">True := Path Point is relative to start point</ >

<"target">Target TCP as pose in meter, angle [x, y, z, roll, pitch, jaw]</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Acceleration in m/s/s</ >

<"v">Velocity in m/s</ >

<"r">

Blend radius in m

Requires AdvancedMode to properly function

</ >

<"mode">

Mode for orientation

0 := Unconstrained mode. Interpolate orientation from current pose to target pose (pose\_to)

1 := Fixed mode. Keep orientation constant relative to the tangent of the circular arc (starting from current pose)

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1 <para />

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result MoveC(double[] poseVia, bool relativeVia, double[] target, bool waitComplete = false, bool angleInRAD = true, double a = 1.0, double v = 0.1, double r = 0.0, int mode = 1, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

### SpeedL

Accelerate in linear in tool-space will let the robot move linear from operator view. The movement required on each joint might be higher.

Accelerate linear in tool-space

<"toolSpeed">Tool speed [m/s] [rad/s] or [deg/s] => [x, y, z, rx, ry, rz] (spatial vector)</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Tool position acceleration in m/s/s</ >

<"t">Time in s before function returns, even if target speed not reached</ >

<"aRot">

Optional: Tool acceleration [rad/s/s]

If not defined a, position acceleration, is used

</ >

<returns>Result object</returns>

public Result SpeedL(double[] toolSpeed, bool waitComplete = false, bool angleInRAD = true, double a = 0.5, double t = 0.5, double aRot = 0.0)

Accelerate in tool-space, move while condition is true

<"toolSpeed">Tool speed [m/s] [rad/s] or [deg/s] => [x, y, z, rx, ry, rz] (spatial vector)</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Tool position acceleration in m/s/s</ >

<"t">Time in s before function returns, even if target speed not reached</ >

<"aRot">

Optional: Tool acceleration [rad/s/s]

If not defined a, position acceleration, is used

</ >

<"source">Select input source: tool-di, std-di, tool-ai, std-ai</ >

<"bit">Select bit/adc channel 0-based</ >

<"value">Value to compare with

For Analog Input: 0...1

</ >

<"boolean">Value to compare with

For Digital Input: False, True

</ >

<"comparison">How to compare: ==, !=, >, <, >=, <=</ >

<returns>Result object</returns>

public Result SpeedL(double[] toolSpeed, bool waitComplete = false, bool angleInRAD = true, double a = 0.5, double t = 0.5, double aRot = 0.0, string source = "tool-di", int bit = 0, double value = 0, bool boolean = false, string comparison = "==")

## Stop Functions

If a stop is called it will cause an immediate stop of the current movement. It can cause a safety stop depending on the current speed and deceleration. Please see robot documentation for more details.

Stop linear in joint-space

<"a">Acceleration in rad/s/s</ >

<returns>Result object</returns>

public Result StopJ(double a = 2.0)

Stop linear in tool-space

<"a">Acceleration in m/s/s</ >

<returns>Result object</returns>

public Result StopL(double a = 20.0)

## Special Functions

The following functions are too specific to handle them in other categories.

### Move Wait

“Wait complete” is designed to keep the function called as long as the robot is active and return when the execution ended. Remark: Some feature can cause a deadlock if “Wait complete” is active.

It could also happen that if this function is called but not in time with a actual movement that it will cause a timeout. **For asynchronous operation better use IsProgramRunning property.**

Wait for program execution from not running to running to not running. Can cause

<"sleeptime">Time to sleep between checks</ >

<"timeout">Timeout</ >

<returns>Result object</returns>

public Result MoveWait(int sleeptime = 100, int timeout = 30000)

### Freedrive

Freedrive mode can be enabled / disabled by calling this function. The robot will stay in Freedrive mode as long as it’s not disabled, allowing the operator to have two hands free

Remarks:

* Robot might start to move if target payload and/or TCP offset are not set properly!
* Freedrive requires Advanced Mode which can interfere with a running advanced sequence. Use with care!

Set Freedrive Mode

Requires Advanced Mode.

Will turn on Advanced Mode if not enabled and restore previous state at end.

<"active">

True := Robot can move freely in all directions.

False := Set robot back in normal position control mode after freedrive mode.

</ >

<returns>Result object</returns>

public Result FreedriveMode(bool active)

### SendScriptFile

SendScriptFile will transfer a UR Scipt File to the robot controller and execute it. The robot will stay active as long as the file gets executed.

Remarks:

* The script file must be compliant with the UR Script File format.

Send a script file to robot over SocktCtrl port.

This function is not compatible to AdvancedMode operation.

<"scriptPath">UR script path</ >

<"isAbsolutePath">True := Path is absolute</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<returns>Result object</returns>

public Result SendScriptFile(string scriptPath, bool isAbsolutePath = true, bool waitComplete = false

### MoveUntilContact

“MoveUntilContact” is a function to detect an obstacle without causing a collision stop in Controller. This function is very delicate in some direction depending on the pose.

As the core part of this function uses the “speedl” behavior it expects tool speed parameters set as speed and direction.

A direction defines where a possible obstacle is expected. All ”0.0000” will allow any contact to stop the movement. This includes a maximum distance from starting point to stop if nothing was detected. The calculation for stop is based on the 3D distance independent from the direction to keep calculations at a minimum. This means that a rotation with no x,y,z-move will not properly work.

Move robot until it comes in contact with any object or when travel exceeds maximum length.

This function is not compatible to AdvancedMode operation.

<"toolSpeed">Tool speed [m/s] [rad/s] or [deg/s] => [x, y, z, rx, ry, rz] (spatial vector)</ >

<"toolDirection">Tool direction [m/s] [rad/s] or [deg/s] => [x, y, z, rx, ry, rz] (spatial vector) and maximum distance it is allowed to travel</ >

<"waitComplete">

True := Wait until movement has completed

False := Just start movement and return

</ >

<"angleInRAD">

True := Angles are in RAD

False := Angles are in DEG

</ >

<"a">Tool position acceleration in m/s/s</ >

<"da">Tool position deceleration in m/s/s</ >

<returns>Result object</returns>

public Result MoveUntilContact(double[] toolSpeed, double[] toolDirection, bool waitComplete = false, bool angleInRAD = true, double a = 0.15, double da = 3)

## Sending Commands(Expert use only)

Commands can be send directly to the robot controller, bypassing all of the logic integrated in SocketCtrl. This should only be used if you know what you do!

### SendCommand (Default to Port 30002)

Normal Mode: Executing this command will take all commands from command list List<string> cmd and wrap them into a robot program before it gets send to robot controller. The Result object returns the formatted string.

Advanced Mode: If SocketCtrl is set into Advanced Mode then the commands from List<string> cmd will be added to taskList.AddRange(cmd); for later execution. The Result object returns the whole taskList as string.

Expert Use Only: Send Robot Command to client socket.

public Result SendCommand(List<string> cmd)

### SendCommandInt (Default to Port 30020)

This function works in Advanced Mode only and uses the interpreter port of the robot controller. All commands from List<string> cmd must be fully formatted including a function header and end.

Expert Use Only: Send Robot Program to client socket interpreter.

public Result SendCommandInt(List<string> cmd)

## Advanced Mode (Expert use only)

Advance Mode defines an extended way of sending commands to the robot that get executed as one robot program. All steps / actions are recorded in a list of strings. This can also be used to execute functions not provided by SocketCtrl. Script functions with indentation (loops, if, else,…) require at least one space char for the indented section. The programmer has to take care of this! If the formatting is wrong then the script will be rejected by the robot controller without any further action.

### Functions

Enable advanced mode

public Result AdvancedModeEnable()

Exceute task list, clear if sent by default

public Result AdvancedModeExecute(bool clear = true)

Exit advanced mode

public Result AdvancedModeExit()

### Properties

Check for advanced mode

public bool IsAdvancedMode { get { return advancedMode; } }

Get Task List by order

public List<string> TaskListGet { get { return taskList; } }

Get Task List by order as single string block

public string TaskListGetString { get { return [String.Format("{0}\n", s);]; } }

Expert Use Only: Append special item to Task List if not available from feature set

public string TaskListAdd { set { taskList.Add(value); } }

Expert Use Only: Clear Task List

public void TaskListClear()