



**UNIVERSAL ROBOTS**

# **The URScript Programming Language**

Version 5.6  
January 16, 2020

The information contained herein is the property of Universal Robots A/S and shall not be reproduced in whole or in part without prior written approval of Universal Robots A/S. The information herein is subject to change without notice and should not be construed as a commitment by Universal Robots A/S. This manual is periodically reviewed and revised.

Universal Robots A/S assumes no responsibility for any errors or omissions in this document.

Copyright © 2009–2020 by Universal Robots A/S

The Universal Robots logo is a registered trademark of Universal Robots A/S.

## Contents

<b>Contents</b>	<b>2</b>
<b>1 The URScript Programming Language</b>	<b>3</b>
1.1 Introduction . . . . .	3
1.2 Connecting to URControl . . . . .	3
1.3 Numbers, Variables, and Types . . . . .	4
1.4 Flow of Control . . . . .	5
1.4.1 Special keywords . . . . .	5
1.5 Function . . . . .	5
1.6 Remote Procedure Call (RPC) . . . . .	6
1.7 Scoping rules . . . . .	7
1.8 Threads . . . . .	9
1.8.1 Threads and scope . . . . .	10
1.8.2 Thread scheduling . . . . .	11
1.9 Program Label . . . . .	11
1.10 Secondary Programs . . . . .	11
<b>2 Module motion</b>	<b>13</b>
2.1 Functions . . . . .	14
2.2 Variables . . . . .	41
<b>3 Module internals</b>	<b>42</b>
3.1 Functions . . . . .	42
3.2 Variables . . . . .	61
<b>4 Module urmath</b>	<b>62</b>
4.1 Functions . . . . .	62
4.2 Variables . . . . .	77
<b>5 Module interfaces</b>	<b>78</b>
5.1 Functions . . . . .	78
5.2 Variables . . . . .	112

<b>6</b>	<b>Module ioconfiguration</b>	<b>113</b>
6.1	Functions . . . . .	113
6.2	Variables . . . . .	120
<b>7</b>	<b>Module processpath</b>	<b>120</b>
7.1	Functions . . . . .	121
7.2	Variables . . . . .	126
7.3	Examples . . . . .	126
7.3.1	Example 1 - Executing a remote TCP path . . . . .	126
7.3.2	Example 2 - Executing a regular TCP path using 5-DOF . . . . .	127
7.3.3	Example 3 - Executing a path and set digital output . . . . .	128
7.4	myToolPathFile.nc . . . . .	129

# 1 The URScript Programming Language

## 1.1 Introduction

The Universal Robot can be controlled at two levels:

- The PolyScope or the Graphical User Interface Level
- Script Level

At the **Script Level**, the **URScript** is the programming language that controls the robot. The **URScript** includes variables, types, and the flow control statements. There are also built-in variables and functions that monitor and control I/O and robot movements.

## 1.2 Connecting to URControl

URControl is the low-level robot controller running on the Mini-ITX PC in the Control Box. When the PC boots up, the URControl starts up as a daemon (i.e., a service) and the PolyScope or Graphical User Interface connects as a client using a local TCP/IP connection.

Programming a robot at the *Script Level* is done by writing a client application (running at another PC) and connecting to URControl using a TCP/IP socket.

- **hostname**: ur-xx (or the IP address found in the **About Dialog-Box** in PolyScope if the robot is not in DNS).
- **port**: 30002

When a connection has been established URScript programs or commands are sent in clear text on the socket. Each line is terminated by "\n". Note that the text can only consist of extended ASCII characters.

The following conditions must be met to ensure that the URControl correctly recognizes the script:

- The script must start from a function definition or a secondary function definition (either "def" or "sec" keywords) in the first column
- All other script lines must be indented by at least one white space
- The last line of script must be an "end" keyword starting in the first column

### 1.3 Numbers, Variables, and Types

In **URScript** arithmetic expression syntax is standard:

```
1+2-3
4*5/6
(1+2)*3/(4-5)
"Hello" + ", " + "World!"
```

In boolean expressions, boolean operators are spelled out:

```
True or False and (1 == 2)
1 > 2 or 3 != 4 xor 5 < -6
not 42 >= 87 and 87 <= 42
"Hello" != "World" and "abc" == "abc"
```

Variable assignment is done using the equal sign =:

```
foo = 42
bar = False or True and not False
baz = 87-13/3.1415
hello = "Hello, World!"
l = [1,2,4]
target = p[0.4,0.4,0.0,0.0,3.14159,0.0]
```

The fundamental type of a variable is deduced from the first assignment of the variable. In the example above `foo` is an `int` and `bar` is a `bool`. `target` is a pose: a combination of a position and orientation.

The fundamental types are:

- `none`
- `bool`
- `number` - either `int` or `float`
- `pose`
- `string`

A pose is given as `p[x, y, z, ax, ay, az]`, where `x, y, z` is the position of the TCP, and `ax, ay, az` is the orientation of the TCP, given in axis-angle notation.

Note that strings are fundamentally byte arrays without knowledge of the encoding used for the characters it contains. Therefore some string functions that may appear to operate on characters (e.g. `str_len`), actually operates on bytes and the result may not correspond to the expected one in case of string containing sequences of multi-byte or variable-length characters. Refer to the description of the single function for more details.

## 1.4 Flow of Control

The flow of control of a program is changed by `if`-statements:

```
if a > 3:
    a = a + 1
elif b < 7:
    b = b * a
else:
    a = a + b
end
```

and `while`-loops:

```
l = [1,2,3,4,5]
i = 0
while i < 5:
    l[i] = l[i]*2
    i = i + 1
end
```

You can use `break` to stop a loop prematurely and `continue` to pass control to the next iteration of the nearest enclosing loop.

### 1.4.1 Special keywords

- `halt` terminates the program.
- `return` returns from a function. When no value is returned, the keyword `None` must follow the keyword `return`.

## 1.5 Function

A function is declared as follows:

```
def add(a, b):
    return a+b
end
```

The function can then be called like this:

```
result = add(1, 4)
```

It is also possible to give function arguments default values:

```
def add(a=0,b=0):
    return a+b
end
```

If default values are given in the declaration, arguments can be either input or skipped as below:

```
result = add(0,0)
result = add()
```

When calling a function, it is important to comply with the declared order of the arguments. If the order is different from its definition, the function does not work as expected.

Arguments can only be passed by value (including arrays). This means that any modification done to the content of the argument within the scope of the function will not be reflected outside that scope.

```
def myProg()

    a = [50,100]

    fun(a)

    def fun(p1):
        p1[0] = 25
        assert(p1[0] == 25)
        ...
    end

    assert(a[0] == 50)
    ...
end
```

URScript also supports named parameters.

## 1.6 Remote Procedure Call (RPC)

**Remote Procedure Calls (RPC)** are similar to normal function calls, except that the function is defined and executed remotely. On the remote site, the **RPC** function being called must exist with the same number of parameters and corresponding types (together the function's signature). If the function is not defined remotely, it stops program execution. The controller uses the XMLRPC standard to send the parameters to the remote site and retrieve the result(s). During an **RPC** call, the controller waits for the remote function to complete. The XMLRPC standard is among others supported by C++ (xmlrpc-c library), Python and Java.

Creating a URScript program to initialize a camera, take a snapshot and retrieve a new target pose:

```
camera = rpc_factory("xmlrpc", "http://127.0.0.1/RPC2")
if (! camera.initialize("RGB")):
    popup("Camera was not initialized")
camera.takeSnapshot()
```

```
target = camera.getTarget()
...
```

First the `rpc_factory` (see `Interfaces` section) creates an XMLRPC connection to the specified **remote** server. The `camera` variable is the handle for the remote function calls. You must initialize the camera and therefore call `camera.initialize("RGB")`. The function returns a boolean value to indicate if the request was successful. In order to find a target position, the camera first takes a picture, hence the `camera.takeSnapshot()` call. Once the snapshot is taken, the image analysis in the remote site calculates the location of the target. Then the program asks for the exact target location with the function call `target = camera.getTarget()`. On return the `target` variable is assigned the result. The `camera.initialize("RGB")`, `takeSnapshot()` and `getTarget()` functions are the responsibility of the RPC server.

The `Technical support website` contains more examples of XMLRPC servers.

## 1.7 Scoping rules

A URScript program is declared as a function without parameters:

```
def myProg():
...
end
```

Every variable declared inside a program has a scope. The scope is the textual region where the variable is directly accessible. Two qualifiers are available to modify this visibility:

- `local` qualifier tells the controller to treat a variable inside a function, as being truly local, even if a global variable with the same name exists.
- `global` qualifier forces a variable declared inside a function, to be globally accessible.

For each variable the controller determines the scope binding, i.e. whether the variable is global or local. In case no `local` or `global` qualifier is specified (also called a free variable), the controller will first try to find the variable in the globals and otherwise the variable will be treated as local.

In the following example, the first `a` is a global variable and the second `a` is a local variable. Both variables are independent, even though they have the same name:

```
def myProg():
...
    global a = 0
...

    def myFun():
        local a = 1
        ...
    end
```



```
...
end
```

Beware that the global variable is no longer accessible from within the function, as the local variable masks the global variable of the same name.

In the following example, the first `a` is a global variable, so the variable inside the function is the same variable declared in the program:

```
def myProg():

    global a = 0

    def myFun():
        a = 1
        ...
    end
    ...
end
```

For each nested function the same scope binding rules hold. In the following example, the first `a` is global defined, the second local and the third implicitly global again:

```
def myProg():

    global a = 0

    def myFun():
        local a = 1

        def myFun2():
            a = 2
            ...
        end
        ...
    end
    ...
end
```

The first and third `a` are one and the same, the second `a` is independent.

Variables on the first scope level (first indentation) are treated as global, even if the `global` qualifier is missing or the `local` qualifier is used:

```
def myProg():

    a = 0

    def myFun():
        a = 1
        ...
    end
```

```
end
...
end
```

The variables `a` are one and the same.

## 1.8 Threads

Threads are supported by a number of special commands.

To declare a new thread a syntax similar to the declaration of functions are used:

```
thread myThread():
  # Do some stuff
  return False
end
```

A couple of things should be noted. First of all, a thread cannot take any parameters, and so the parentheses in the declaration must be empty. Second, although a return statement is allowed in the thread, the value returned is discarded, and cannot be accessed from outside the thread. A thread can contain other threads, the same way a function can contain other functions. Threads can in other words be nested, allowing for a thread hierarchy to be formed.

To run a thread use the following syntax:

```
thread myThread():
  # Do some stuff
  return False
end

thrd = run myThread()
```

The value returned by the `run` command is a handle to the running thread. This handle can be used to interact with a running thread. The `run` command spawns from the new thread, and then executes the instruction following the `run` instruction.

A thread can only wait for a running thread spawned by itself. To wait for a running thread to finish, use the `join` command:

```
thread myThread():
  # Do some stuff
  return False
end
```

```
thrd = run myThread()

join thrd
```

This halts the calling threads execution, until the specified thread finishes its execution. If the thread is already finished, the statement has no effect.

To kill a running thread, use the `kill` command:

```
thread myThread():
    # Do some stuff
    return False
end

thrd = run myThread()

kill thrd
```

After the call to `kill`, the thread is stopped, and the thread handle is no longer valid. If the thread has children, these are killed as well.

To protect against race conditions and other thread-related issues, support for critical sections is provided. A critical section ensures the enclosed code can finish running before another thread can start running. The previous statement is always true, unless a time-demanding command is present within the scope of the critical section. In such a case, another thread will be allowed to run. Time-demanding commands include `sleep`, `sync`, `move-commands`, and `socketRead`. Therefore, it is important to keep the critical section as short as possible. The syntax is as follows:

```
thread myThread():
    enter_critical
    # Do some stuff
    exit_critical
    return False
end
```

### 1.8.1 Threads and scope

The scoping rules for threads are exactly the same, as those used for functions. See 1.7 for a discussion of these rules.

### 1.8.2 Thread scheduling

Because the primary purpose of the URScript scripting language is to control the robot, the scheduling policy is largely based upon the realtime demands of this task.

The robot must be controlled a frequency of 500 Hz, or in other words, it must be told what to do every 0.002 second (each 0.002 second period is called a frame). To achieve this, each thread is given a “physical” (or robot) time slice of 0.002 seconds to use, and all threads in a runnable state is then scheduled in a round robin<sup>1</sup> fashion.

Each time a thread is scheduled, it can use a piece of its time slice (by executing instructions that control the robot), or it can execute instructions that do not control the robot, and therefore do not use any “physical” time. If a thread uses up its entire time slice, either by use of “physical” time or by computational heavy instructions (such as an infinite loop that do not control the robot) it is placed in a non-runnable state, and is not allowed to run until the next frame starts. If a thread does not use its time slice within a frame, it is expected to switch to a non-runnable state before the end of the frame<sup>2</sup>. The reason for this state switching can be a join instruction or simply because the thread terminates.

It should be noted that even though the `sleep` instruction does not control the robot, it still uses “physical” time. The same is true for the `sync` instruction. Inserting `sync` or `sleep` will allow time for other threads to be executed and is therefore recommended to use next to computational heavy instructions or inside infinite loops that do not control the robot, otherwise an exception like “Runtime is too much behind” can be raised with a consequent protective stop.

## 1.9 Program Label

Program label code lines, with an “\$” as first symbol, are special lines in programs generated by PolyScope that make it possible to track the execution of a program.

```
$ 2 "var_1= True "
global var_1= True
```

### 1.10 Secondary Programs

Secondary program is executed by the robot controller concurrently and simultaneously with the primary script program. It could be used to handle the I/O signals, while the primary program moves the robot between waypoints. A secondary program could be sent to controller via primary or secondary TCP/IP socket, just like any other script program and must follow the same script syntax as regular robot programs.

Notable exception is that secondary program should not use any “physical” time. In

<sup>1</sup>Before the start of each frame the threads are sorted, such that the thread with the largest remaining time slice is to be scheduled first.

<sup>2</sup>If this expectation is not met, the program is stopped.

particular, it cannot contain `sleep`, or `move` statements. Secondary program must be simple enough to be executed in a single controller step, without blocking the realtime thread.

Exceptions on secondary program do not stop execution of the primary program. Exceptions are reported in robot controller log files.

The secondary function must be defined using the keyword `"sec"` as follows:

```
sec secondaryProgram():  
    set_digital_out(1,True)  
end
```

## 2 Module motion

This module contains functions and variables built into the URScript programming language.

URScript programs are executed in real-time in the URControl RuntimeMachine (RTMachine). The RuntimeMachine communicates with the robot with a frequency of 500hz.

Robot trajectories are generated online by calling the move functions `movej`, `movel` and the speed functions `speedj`, `speedl`.

Joint positions ( $q$ ) and joint speeds ( $q\dot{d}$ ) are represented directly as lists of 6 Floats, one for each robot joint. Tool poses ( $x$ ) are represented as poses also consisting of 6 Floats. In a pose, the first 3 coordinates is a position vector and the last 3 an axis-angle ([http://en.wikipedia.org/wiki/Axis\\_angle](http://en.wikipedia.org/wiki/Axis_angle)).

## 2.1 Functions

### **conveyor\_pulse\_decode**(type, A, B)

*Deprecated:* Tells the robot controller to treat digital inputs number A and B as pulses for a conveyor encoder. Only digital input 0, 1, 2 or 3 can be used.

#### **Parameters**

**type:** An integer determining how to treat the inputs on A and B

0 is no encoder, pulse decoding is disabled.

1 is quadrature encoder, input A and B must be square waves with 90 degree offset. Direction of the conveyor can be determined.

2 is rising and falling edge on single input (A).

3 is rising edge on single input (A).

4 is falling edge on single input (A).

The controller can decode inputs at up to 40kHz

**A:** Encoder input A, values of 0-3 are the digital inputs 0-3.

**B:** Encoder input B, values of 0-3 are the digital inputs 0-3.

**Deprecated:** This function is replaced by `encoder_enable_pulse_decode` and it should therefore not be used moving forward.

```
>>> conveyor_pulse_decode(1, 0, 1)
```

This example shows how to set up quadrature pulse decoding with input A = `digital_in(0)` and input B = `digital_in(1)`

```
>>> conveyor_pulse_decode(2, 3)
```

This example shows how to set up rising and falling edge pulse decoding with input A = `digital_in(3)`. Note that you do not have to set parameter B (as it is not used anyway).

**Example command:** `conveyor_pulse_decode(1, 2, 3)`

- Example Parameters:
  - type = 1 → is quadrature encoder, input A and B must be square waves with 90 degree offset. Direction of the conveyor can be determined.
  - A = 2 → Encoder output A is connected to digital input 2
  - B = 3 → Encoder output B is connected to digital input 3

**encoder\_enable\_pulse\_decode(encoder\_index, decoder\_type, A, B)**

Sets up an encoder hooked up to a pulse decoder of the controller.

```
>>> encoder_enable_pulse_decode(0, 0, 1, 8, 9)
```

This example shows how to set up encoder 0 decoding a quadrature signal connected to pin 8 and 9.

**Parameters**

**encoder\_index:** Index of the encoder to define. Must be either 0 or 1.

**decoder\_type:** An integer determining how to treat the inputs on A and B.

0 is no encoder, pulse decoding is disabled.

1 is quadrature encoder, input A and B must be square waves with 90 degree offset. Direction of the conveyor can be determined.

2 is rising and falling edge on single input (A).

3 is rising edge on single input (A).

4 is falling edge on single input (A).

The controller can decode inputs at up to 40kHz

**A:** Encoder input A pin. Must be 8-11.

**B:** Encoder input B pin. Must be 8-11.



**encoder\_enable\_set\_tick\_count(encoder\_index, range\_id)**

Sets up an encoder expecting to be updated with tick counts via the function `encoder_set_tick_count`.

```
>>> encoder_enable_set_tick_count(0, 0)
```

This example shows how to set up encoder 0 to expect counts in the range of (-2147483648 ; 2147483647).

**Parameters**

`encoder_index`: Index of the encoder to define. Must be either 0 or 1.

`range_id`: decoder\_index: Range of the encoder (integer). Needed to handle wrapping nicely.

0 is a 32 bit signed encoder, range (-2147483648 ; 2147483647)

1 is a 8 bit unsigned encoder, range (0 ; 255)

2 is a 16 bit unsigned encoder, range (0 ; 65535)

3 is a 24 bit unsigned encoder, range (0 ; 16777215)

4 is a 32 bit unsigned encoder, range (0 ; 4294967295)

**encoder\_get\_tick\_count(encoder\_index)**

Returns the tick count of the designated encoder.

```
>>> encoder_get_tick_count(0)
```

This example returns the current tick count of encoder 0. Use caution when subtracting encoder tick counts. Please see the function `encoder_unwind_delta_tick_count`.

**Parameters**

`encoder_index`: Index of the encoder to query. Must be either 0 or 1.

**Return Value**

The conveyor encoder tick count (float)

**encoder\_set\_tick\_count**(*encoder\_index*, *count*)

Tells the robot controller the tick count of the encoder. This function is useful for absolute encoders (e.g. MODBUS).

```
>>> encoder_set_tick_count(0, 1234)
```

This example sets the tick count of encoder 0 to 1234. Assumes that the encoder is enabled using `encoder_enable_set_tick_count` first.

**Parameters**

<code>encoder_index</code> :	Index of the encoder to define. Must be either 0 or 1.
<code>count</code> :	The tick count to set. Must be within the range of the encoder.

**encoder\_unwind\_delta\_tick\_count(encoder\_index, delta\_tick\_count)**

Returns the delta\_tick\_count. Unwinds in case encoder wraps around the range. If no wrapping has happened the given delta\_tick\_count is returned without any modification.

Consider the following situation: You are using an encoder with a UINT16 range, meaning the tick count is always in the (0; 65536( range. When the encoder is ticking, it may cross either end of the range, which causes the tick count to wrap around to the other end. During your program, the current tick count is assigned to a variable (start:=encoder\_get\_tick\_count(...)). Later, the tick count is assigned to another variable (current:=encoder\_get\_tick\_count(...)). To calculate the distance the conveyor has traveled between the two sample points, the two tick counts are subtracted from each other.

For example, the first sample point is near the end of the range (e.g., start:=65530). When the conveyor arrives at the second point, the encoder may have crossed the end of its range, wrapped around, and reached a value near the beginning of the range (e.g., current:=864). Subtracting the two samples to calculate the motion of the conveyor is not robust, and may result in an incorrect result (delta=current-start=-64666).

Conveyor tracking applications rely on these kinds of encoder calculations. Unless special care is taken to compensate the encoder wrapping around, the application will not be robust and may produce weird behaviors (e.g., singularities or exceeded speed limits) which are difficult to explain and to reproduce.

This heuristic function checks that a given delta\_tick\_count value is reasonable. If the encoder wrapped around the end of the range, it compensates (i.e., unwinds) and returns the adjusted result. If a delta\_tick\_count is larger than half the range of the encoder, wrapping is assumed and is compensated. As a consequence, this function only works when the range of the encoder is explicitly known, and therefore the designated encoder must be enabled. If not, this function will always return nil.

**Parameters**

`encoder_index`: Index of the encoder to query. Must be either 0 or 1.

`delta_tick_count`: The delta (difference between two) tick count to unwind (float)

**Return Value**

The unwound delta\_tick\_count (float)

**end\_force\_mode()**

Resets the robot mode from force mode to normal operation.

This is also done when a program stops.

**end\_freedrive\_mode()**

Set robot back in normal position control mode after freedrive mode.

**end\_screw\_driving()**

Exit screw driving mode and return to normal operation.

**end\_teach\_mode()**

Set robot back in normal position control mode after freedrive mode.

**force\_mode(task\_frame, selection\_vector, wrench, type, limits)**

Set robot to be controlled in force mode

**Parameters**

<code>task_frame:</code>	A pose vector that defines the force frame relative to the base frame.
<code>selection_vector:</code>	A 6d vector of 0s and 1s. 1 means that the robot will be compliant in the corresponding axis of the task frame.
<code>wrench:</code>	<p>The forces/torques the robot will apply to its environment. The robot adjusts its position along/about compliant axis in order to achieve the specified force/torque. Values have no effect for non-compliant axes.</p> <p>Actual wrench applied may be lower than requested due to joint safety limits. Actual forces and torques can be read using <code>get_tcp_force</code> function in a separate thread.</p>
<code>type:</code>	<p>An integer (1;3) specifying how the robot interprets the force frame.</p> <p>1: The force frame is transformed in a way such that its y-axis is aligned with a vector pointing from the robot tcp towards the origin of the force frame.</p> <p>2: The force frame is not transformed.</p> <p>3: The force frame is transformed in a way such that its x-axis is the projection of the robot tcp velocity vector onto the x-y plane of the force frame.</p>
<code>limits:</code>	(Float) 6d vector. For compliant axes, these values are the maximum allowed tcp speed along/about the axis. For non-compliant axes, these values are the maximum allowed deviation along/about an axis between the actual tcp position and the one set by the program.

**Note:** Remember to tare the force-torque sensor by calling `zero_ftsensor()` first. Avoid movements parallel to compliant axes and high deceleration (consider inserting a short sleep command of at least 0.02s) just before entering force mode. Avoid high acceleration in force mode as this decreases the force control accuracy.

**force\_mode\_example()**

This is an example of the above `force_mode()` function

**Example command:** `force_mode(p[0.1,0,0,0,0.785],  
[1,0,0,0,0,0], [20,0,40,0,0,0], 2,  
[.2,.1,.1,.785,.785,1.57])`

- Example Parameters:
  - Task frame = `p(0.1,0,0,0,0.785)` → This frame is offset from the base frame 100 mm in the x direction and rotated 45 degrees in the rz direction
  - Selection Vector = `(1,0,0,0,0)` → The robot is compliant in the x direction of the Task frame above.
  - Wrench = `(20,0,40,0,0)` → The robot applies 20N in the x direction. It also accounts for a 40N external force in the z direction.
  - Type = 2 → The force frame is not transformed.
  - Limits = `(.1,.1,.1,.785,.785,1.57)` → max x velocity is 100 mm/s, max y deviation is 100 mm, max z deviation is 100 mm, max rx deviation is 45 deg, max ry deviation is 45 deg, max rz deviation is 90 deg.

**force\_mode\_set\_damping(*damping*)**

Sets the damping parameter in force mode.

**Parameters**

**damping:** Between 0 and 1, default value is 0.005.

A value of 1 is full damping, so the robot will decelerate quickly if no force is present. A value of 0 is no damping, here the robot will maintain the speed.

The value is stored until this function is called again. Add this to the beginning of your program to ensure it is called before force mode is entered (otherwise default value will be used).

**force\_mode\_set\_gain\_scaling(*scaling*)**

Scales the gain in force mode.

**Parameters**

*scaling*: scaling parameter between 0 and 2, default is 1.

A value larger than 1 can make force mode unstable, e.g. in case of collisions or pushing against hard surfaces.

The value is stored until this function is called again. Add this to the beginning of your program to ensure it is called before force mode is entered (otherwise default value will be used).

**freedrive\_mode()**

Set robot in freedrive mode. In this mode the robot can be moved around by hand in the same way as by pressing the "freedrive" button. The robot will not be able to follow a trajectory (eg. a movej) in this mode.

**get\_conveyor\_tick\_count()**

*Deprecated*: Tells the tick count of the encoder, note that the controller interpolates tick counts to get more accurate movements with low resolution encoders

**Return Value**

The conveyor encoder tick count

**Deprecated**: This function is replaced by `encoder_get_tick_count` and it should therefore not be used moving forward.

**get\_target\_tcp\_pose\_along\_path()**

Query the target TCP pose as given by the trajectory being followed.

This script function is useful in conjunction with the path offsetting feature or conveyor tracking to know what the target pose of the TCP would be if no offset was applied.

**Return Value**

Target TCP pose

**get\_target\_tcp\_speed\_along\_path()**

Query the target TCP speed as given by the trajectory being followed.

This script function is useful in conjunction with the path offsetting feature and conveyor tracking to know what the target speed of the TCP would be if no offset was applied.

**Return Value**

Target TCP speed as a vector



---

**movec**(pose\_via, pose\_to, a=1.2, v=0.25, r=0, mode=0)
 

---

Move Circular: Move to position (circular in tool-space)

TCP moves on the circular arc segment from current pose, through pose\_via to pose\_to. Accelerates to and moves with constant tool speed v. Use the mode parameter to define the orientation interpolation.

### Parameters

- pose\_via: path point (note: only position is used). Pose\_via can also be specified as joint positions, then forward kinematics is used to calculate the corresponding pose.
- pose\_to: target pose (note: only position is used in Fixed orientation mode). Pose\_to can also be specified as joint positions, then forward kinematics is used to calculate the corresponding pose.
- a: tool acceleration (m/s<sup>2</sup>)
- v: tool speed (m/s)
- r: blend radius (of target pose) (m)
- mode: 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)

**Example command:** movec(p[x,y,z,0,0,0], pose\_to, a=1.2, v=0.25, r=0.05, mode=1)

- Example Parameters:
  - Note: first position on circle is previous waypoint.
  - pose\_via = p(x,y,z,0,0,0) → second position on circle.
    - \* Note rotations are not used so they can be left as zeros.
    - \* Note: This position can also be represented as joint angles (j0,j1,j2,j3,j4,j5) then forward kinematics is used to calculate the corresponding pose
  - pose\_to → third (and final) position on circle
  - a = 1.2 → acceleration is 1.2 m/s/s
  - v = 0.25 → velocity is 250 mm/s
  - r = 0 → blend radius (at pose\_to) is 50 mm.
  - mode = 1 → use fixed orientation relative to tangent of circular arc

**movej**( $q, a=1.4, v=1.05, t=0, r=0$ )

Move to position (linear in joint-space)

When using this command, the robot must be at a standstill or come from a movej or movel with a blend. The speed and acceleration parameters control the trapezoid speed profile of the move. Alternatively, the  $t$  parameter can be used to set the time for this move. Time setting has priority over speed and acceleration settings.

#### Parameters

- $q$ : joint positions ( $q$  can also be specified as a pose, then inverse kinematics is used to calculate the corresponding joint positions)
- $a$ : joint acceleration of leading axis (rad/s<sup>2</sup>)
- $v$ : joint speed of leading axis (rad/s)
- $t$ : time (s)
- $r$ : blend radius (m)

If a blend radius is set, the robot arm trajectory will be modified to avoid the robot stopping at the point.

However, if the blend region of this move overlaps with the blend radius of previous or following waypoints, this move will be skipped, and an 'Overlapping Blends' warning message will be generated.

**Example command:** `movej([0, 1.57, -1.57, 3.14, -1.57, 1.57], a=1.4, v=1.05, t=0, r=0)`

- Example Parameters:
  - $q = (0, 1.57, -1.57, 3.14, -1.57, 1.57)$  → base is at 0 deg rotation, shoulder is at 90 deg rotation, elbow is at -90 deg rotation, wrist 1 is at 180 deg rotation, wrist 2 is at -90 deg rotation, wrist 3 is at 90 deg rotation. Note: joint positions ( $q$  can also be specified as a pose, then inverse kinematics is used to calculate the corresponding joint positions)
  - $a = 1.4$  → acceleration is 1.4 rad/s/s
  - $v = 1.05$  → velocity is 1.05 rad/s
  - $t = 0$  → the time (seconds) to make move is not specified. If it were specified the command would ignore the  $a$  and  $v$  values.
  - $r = 0$  → the blend radius is zero meters.

**move1**(pose, a=1.2, v=0.25, t=0, r=0)

Move to position (linear in tool-space)

See movej.

**Parameters**

pose: target pose (pose can also be specified as joint positions, then forward kinematics is used to calculate the corresponding pose)

a: tool acceleration (m/s<sup>2</sup>)

v: tool speed (m/s)

t: time (S)

r: blend radius (m)

**Example command:** move1(pose, a=1.2, v=0.25, t=0, r=0)

- Example Parameters:
  - pose = p(0.2,0.3,0.5,0,0,3.14) → position in base frame of x = 200 mm, y = 300 mm, z = 500 mm, rx = 0, ry = 0, rz = 180 deg
  - a = 1.2 → acceleration of 1.2 m/s<sup>2</sup>
  - v = 0.25 → velocity of 250 mm/s
  - t = 0 → the time (seconds) to make the move is not specified. If it were specified the command would ignore the a and v values.
  - r = 0 → the blend radius is zero meters.

---

**movep**(pose, a=1.2, v=0.25, r=0)

Move Process

Blend circular (in tool-space) and move linear (in tool-space) to position. Accelerates to and moves with constant tool speed v.

#### Parameters

- pose: target pose (pose can also be specified as joint positions, then forward kinematics is used to calculate the corresponding pose)
- a: tool acceleration (m/s<sup>2</sup>)
- v: tool speed (m/s)
- r: blend radius (m)

**Example command:** movep(pose, a=1.2, v=0.25, r=0)

- Example Parameters:
  - pose = p(0.2,0.3,0.5,0,0,3.14) → position in base frame of x = 200 mm, y = 300 mm, z = 500 mm, rx = 0, ry = 0, rz = 180 deg.
  - a = 1.2 → acceleration of 1.2 m/s<sup>2</sup>
  - v = 0.25 → velocity of 250 mm/s
  - r = 0 → the blend radius is zero meters.

---

**path\_offset\_disable**(a=20)

Disable the path offsetting and decelerate all joints to zero speed.

Uses the `stopj` functionality to bring all joints to a rest. Therefore, all joints will decelerate at different rates but reach stand-still at the same time.

Use the script function `path_offset_enable` to enable path offsetting

#### Parameters

- a: joint acceleration (rad/s<sup>2</sup>) (optional)

**path\_offset\_enable()**

Enable path offsetting.

Path offsetting is used to superimpose a Cartesian offset onto the robot motion as it follows a trajectory. This is useful for instance for imposing a weaving motion onto a welding task, or to compensate for the effect of moving the base of the robot while following a trajectory.

Path offsets can be applied in various frames of reference and in various ways. Please refer to the script function `path_offset_set` for further explanation.

Enabling path offsetting doesn't cancel the effects of previous calls to the script functions `path_offset_set_max_offset` and `path_offset_set_alpha_filter`. Path offset configuration will persist through cycles of enable and disable.

**path\_offset\_get(type)**

Query the offset currently applied.

**Parameters**

`type`: Specifies the frame of reference of the returned offset. Please refer to the `path_offset_set` script function for a definition of the possible values and their meaning.

**Return Value**

Pose specifying the translational and rotational offset. Units are meters and radians.

**path\_offset\_set(offset, type)**

Specify the Cartesian path offset to be applied.

Use the script function `path_offset_enable` beforehand to enable offsetting. The calculated offset is applied during each cycle at 500Hz.

Discontinuous or jerky offsets are likely to cause protective stops. If offsets are not smooth the function `path_offset_set_alpha_filter` can be used to engage a simple filter.

The following example uses a harmonic wave (cosine) to offset the position of the TCP along the Z-axis of the robot base:

```
>>> thread OffsetThread():
>>>     while(True):
>>>         # 2Hz cosine wave with an amplitude of 5mm
>>>         global x = 0.005*(cos(p) - 1)
>>>         global p = p + 4*3.14159/500
>>>         path_offset_set([0,0,x,0,0,0], 1)
>>>         sync()
>>>     end
>>> end
```

**Parameters**

**offset:** Pose specifying the translational and rotational offset.

**type:** Specifies how to apply the given offset. Options are:

- 1: (BASE) Use robot base coordinates when applying.
- 2: (TCP) Use robot TCP coordinates when applying.
- 3: (MOTION) Use a coordinate system following the un-offset trajectory when applying. This coordinate system is defined as follows. X-axis along the tangent of the translational part of the un-offset trajectory (rotation not relevant here). Y-axis perpendicular to the X-axis above and the Z-axis of the tool (X cross Z). Z-axis given from the X and Y axes by observing the right-hand rule. This is useful for instance for superimposing a weaving pattern onto the trajectory when welding.
- 4: (BASE INVERTED) Invert the offset and use robot base coordinates when applying. This is useful for compensation when the robot base is moving i.e. keeping the robot TCP steady when the base is offset.

**path\_offset\_set\_alpha\_filter(*alpha*)**

Engage offset filtering using a simple alpha filter (EWMA) and set the filter coefficient.

An alpha filter is a very simple 1st order IIR filter using a weighted sum of the target and current offset:  $\text{current} = \alpha * \text{target} + (1 - \alpha) * \text{current}$

**Parameters**

**alpha:** The filter coefficient to be used. Must be between 0 and 1. A value of 1 is equivalent to no filtering. For welding, experiments have shown that a value around 0.1 is a good compromise between robustness and offsetting accuracy.

**path\_offset\_set\_max\_offset(*transLimit*, *rotLimit*)**

Set limits for the maximum allowed offset.

Due to safety and due to the finite reach of the robot, path offsetting limits the magnitude of the offset to be applied. Use this function to adjust these limits. Per default limits of 0.1 meters and 30 degrees (0.52 radians) are used.

**Parameters**

**transLimit:** The maximum allowed translational offset distance along any axis in meters.

**rotLimit:** The maximum allowed rotational offset around any axis in radians.

**position\_deviation\_warning**(*enabled*, *threshold*=0.8)

When enabled, this function generates warning messages to the log when the robot deviates from the target position. This function can be called at any point in the execution of a program. It has no return value.

```
>>> position_deviation_warning(True)
```

In the above example, the function has been enabled. This means that log messages will be generated whenever a position deviation occurs. The optional "threshold" parameter can be used to specify the level of position deviation that triggers a log message.

**Parameters**

- enabled:** (Boolean) Enable or disable position deviation log messages.
- threshold:** (Float) Optional value in the range (0;1), where 0 is no position deviation and 1 is the maximum position deviation (equivalent to the amount of position deviation that causes a protective stop of the robot). If no threshold is specified by the user, a default value of 0.8 is used.

**Example command:** `position_deviation_warning(True, 0.8)`

- Example Parameters:
  - Enabled = True → Logging of warning is turned on
  - Threshold = 0.8 → 80% of deviation that causes a protective stop causes a warning to be logged in the log history file.



**reset\_revolution\_counter**(qNear=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0])

Reset the revolution counter, if no offset is specified. This is applied on joints which safety limits are set to "Unlimited" and are only applied when new safety settings are applied with limited joint angles.

```
>>> reset_revolution_counter()
```

**Parameters**

qNear: Optional parameter, reset the revolution counter to one close to the given qNear joint vector. If not defined, the joint's actual number of revolutions are used.

**Example command:** `reset_revolution_counter(qNear=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0])`

- Example Parameters:
  - qNear = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0) → Optional parameter, resets the revolution counter of wrist 3 to zero on UR3 robots to the nearest zero location to joint rotations represented by qNear.

**screw\_driving**(*f*, *v\_limit*)

Enter screw driving mode. The robot will exert a force in the TCP Z-axis direction at limited speed. This allows the robot to follow the screw during tightening/loosening operations.

**Parameters**

*f*: The amount of force the robot will exert along the TCP Z-axis (Newtons).

*v\_limit*: Maximum TCP velocity along the Z axis (m/s).

**Notes:**

- Zero the F/T sensor without the screw driver pushing against the screw.
- Call `end_screw_driving` when the screw driving operation has completed.

```
>>> def testScrewDriver():
>>>     # Zero F/T sensor
>>>     zero_ftsensor()
>>>     sleep(0.02)
>>>
>>>     # Move the robot to the tightening position
>>>     # (i.e. just before contact with the screw)
>>>     ...
>>>
>>>     # Start following the screw while tightening
>>>     screw_driving(5.0, 0.1)
>>>
>>>     # Wait until screw driver reports OK or NOK
>>>     ...
>>>
>>>     # Exit screw driving mode
>>>     end_screw_driving()
>>> end
```

**servoc**(pose, a=1.2, v=0.25, r=0)

Servo Circular

Servo to position (circular in tool-space). Accelerates to and moves with constant tool speed v.

**Parameters**

- pose: target pose (pose can also be specified as joint positions, then forward kinematics is used to calculate the corresponding pose)
- a: tool acceleration (m/s<sup>2</sup>)
- v: tool speed (m/s)
- r: blend radius (of target pose) (m)

**Example command:** `servoc(p[0.2,0.3,0.5,0,0,3.14], a=1.2, v=0.25, r=0)`

- Example Parameters:
  - pose = p(0.2,0.3,0.5,0,0,3.14) → position in base frame of x = 200 mm, y = 300 mm, z = 500 mm, rx = 0, ry = 0, rz = 180 deg.
  - a = 1.2 → acceleration of 1.2 m/s<sup>2</sup>
  - v = 0.25 → velocity of 250 mm/s
  - r = 0 → the blend radius at the target position is zero meters.

---

**servoj**(*q, a, v, t=0.002, lookahead\_time=0.1, gain=300*)

---

Servoj can be used for online realtime control of joint positions.

The gain parameter works the same way as the P-term of a PID controller, where it adjusts the current position towards the desired (*q*). The higher the gain, the faster reaction the robot will have.

The parameter *lookahead\_time* is used to project the current position forward in time with the current velocity. A low value gives fast reaction, a high value prevents overshoot.

Note: A high gain or a short lookahead time may cause instability and vibrations. Especially if the target positions are noisy or updated at a low frequency

It is preferred to call this function with a new setpoint (*q*) in each time step (thus the default *t=0.002*)

You can combine with the script command `get_inverse_kin()` to perform servoing based on cartesian positions:

```
>>> q = get_inverse_kin(x)
>>> servoj(q, lookahead_time=0.05, gain=500)
```

Here *x* is a pose variable with target cartesian positions, received over a socket or RTDE registers.

**Example command:** `servoj([0.0,1.57,-1.57,0,0,3.14], 0, 0, 0.002, 0.1, 300)`

- Example Parameters:
  - *q* = (0.0,1.57,-1.57,0,0,3.14) → joint angles in radians representing rotations of base, shoulder, elbow, wrist1, wrist2 and wrist3
  - *a* = 0 → not used in current version
  - *v* = 0 → not used in current version
  - *t* = 0.002 → time where the command is controlling the robot. The function is blocking for time *t* (S).
  - lookahead time = .1 → time (S), range (0.03,0.2) smoothes the trajectory with this lookahead time
  - gain = 300 → proportional gain for following target position, range (100,2000)

**set\_conveyor\_tick\_count**(*tick\_count*, *absolute\_encoder\_resolution*=0)

*Deprecated:* Tells the robot controller the tick count of the encoder. This function is useful for absolute encoders, use `conveyor_pulse_decode()` for setting up an incremental encoder. For circular conveyors, the value must be between 0 and the number of ticks per revolution.

#### Parameters

<code>tick_count:</code>	Tick count of the conveyor (Integer)
<code>absolute_encoder_resolution:</code>	Resolution of the encoder, needed to handle wrapping nicely. (Integer)
	0 is a 32 bit signed encoder, range (-2147483648 ; 2147483647) (default)
	1 is a 8 bit unsigned encoder, range (0 ; 255)
	2 is a 16 bit unsigned encoder, range (0 ; 65535)
	3 is a 24 bit unsigned encoder, range (0 ; 16777215)
	4 is a 32 bit unsigned encoder, range (0 ; 4294967295)

**Deprecated:** This function is replaced by `encoder_set_tick_count` and it should therefore not be used moving forward.

**Example command:** `set_conveyor_tick_count (24543, 0)`

- Example Parameters:
  - `Tick_count = 24543` → a value read from e.g. a MODBUS register being updated by the absolute encoder
  - `Absolute_encoder_resolution = 0` → 0 is a 32 bit signed encoder, range (-2147483648 ; 2147483647) (default)

**set\_pos(*q*)**

Set joint positions of simulated robot

**Parameters**

*q*: joint positions

**Example command:** `set_pos([0.0, 1.57, -1.57, 0, 0, 3.14])`

- Example Parameters:
  - $q = (0.0, 1.57, -1.57, 0, 0, 3.14)$  → the position of the simulated robot with joint angles in radians representing rotations of base, shoulder, elbow, wrist1, wrist2 and wrist3

**set\_safety\_mode\_transition\_hardness(*type*)**

Sets the transition hardness between normal mode, reduced mode and safeguard stop.

**Parameters**

*type*: An integer specifying transition hardness.

0 is hard transition between modes using maximum torque, similar to emergency stop.

1 is soft transition between modes.

**speedj(*qd*, *a*, *t*)**

Joint speed

Accelerate linearly in joint space and continue with constant joint speed. The time *t* is optional; if provided the function will return after time *t*, regardless of the target speed has been reached. If the time *t* is not provided, the function will return when the target speed is reached.

**Parameters**

*qd*: joint speeds (rad/s)

*a*: joint acceleration (rad/s<sup>2</sup>) (of leading axis)

*t*: time (s) before the function returns (optional)

**Example command:** `speedj([0.2, 0.3, 0.1, 0.05, 0, 0], 0.5, 0.5)`

- Example Parameters:
  - *qd* → Joint speeds of: base=0.2 rad/s, shoulder=0.3 rad/s, elbow=0.1 rad/s, wrist1=0.05 rad/s, wrist2 and wrist3=0 rad/s
  - *a* = 0.5 rad/s<sup>2</sup> → acceleration of the leading axis (shoulder in this case)
  - *t* = 0.5 s → time before the function returns

**speedl(*xd, a, t, aRot='a'*)**

Tool speed

Accelerate linearly in Cartesian space and continue with constant tool speed. The time *t* is optional; if provided the function will return after time *t*, regardless of the target speed has been reached. If the time *t* is not provided, the function will return when the target speed is reached.

**Parameters**

- xd*: tool speed (m/s) (spatial vector)
- a*: tool position acceleration (m/s<sup>2</sup>)
- t*: time (s) before function returns (optional)
- aRot*: tool acceleration (rad/s<sup>2</sup>) (optional), if not defined *a*, position acceleration, is used

**Example command:** `speedl([0.5,0.4,0,1.57,0,0], 0.5, 0.5)`

- Example Parameters:
  - *xd* → Tool speeds of: x=500 mm/s, y=400 mm/s, rx=90 deg/s, ry and rz=0 mm/s
  - *a* = 0.5 rad/s<sup>2</sup> → acceleration of the leading axis (shoulder is this case)
  - *t* = 0.5 s → time before the function returns

**stop\_conveyor\_tracking(*a=20*)**

Stop tracking the conveyor, started by `track_conveyor_linear()` or `track_conveyor_circular()`, and decelerate all joint speeds to zero.

**Parameters**

- a*: joint acceleration (rad/s<sup>2</sup>) (optional)

**Example command:** `stop_conveyor_tracking(a=15)`

- Example Parameters:
  - *a* = 15 rad/s<sup>2</sup> → acceleration of the joints

**stopj(a)**

Stop (linear in joint space)

Decelerate joint speeds to zero

**Parameters**

a: joint acceleration ( $\text{rad/s}^2$ ) (of leading axis)

**Example command:** `stopj(2)`

- Example Parameters:
  - $a = 2 \text{ rad/s}^2 \rightarrow$  rate of deceleration of the leading axis.

**stopl(a, aRot='a')**

Stop (linear in tool space)

Decelerate tool speed to zero

**Parameters**

a: tool acceleration ( $\text{m/s}^2$ )

aRot: tool acceleration ( $\text{rad/s}^2$ ) (optional), if not defined  
a, position acceleration, is used

**Example command:** `stopl(20)`

- Example Parameters:
  - $a = 20 \text{ m/s}^2 \rightarrow$  rate of deceleration of the tool
  - aRot  $\rightarrow$  tool deceleration ( $\text{rad/s}^2$ ) (optional), if not defined, position acceleration, is used. i.e. it supersedes the "a" deceleration.

**teach\_mode()**

Set robot in freedrive mode. In this mode the robot can be moved around by hand in the same way as by pressing the "freedrive" button. The robot will not be able to follow a trajectory (eg. a movej) in this mode.



**track\_conveyor\_circular**(center, ticks\_per\_revolution,  
rotate\_tool='False', encoder\_index=0)

Makes robot movement (movej() etc.) track a circular conveyor.

```
>>> track_conveyor_circular(p[0.5,0.5,0,0,0,0],500.0, false)
```

The example code makes the robot track a circular conveyor with center in p(0.5,0.5,0,0,0,0) of the robot base coordinate system, where 500 ticks on the encoder corresponds to one revolution of the circular conveyor around the center.

#### Parameters

center:	Pose vector that determines center of the conveyor in the base coordinate system of the robot.
ticks_per_revolution:	How many ticks the encoder sees when the conveyor moves one revolution.
rotate_tool:	Should the tool rotate with the conveyor or stay in the orientation specified by the trajectory (move() etc.).
encoder_index:	The index of the encoder to associate with the conveyor tracking. Must be either 0 or 1. This is an optional argument, and please note the default of 0. The ability to omit this argument will allow existing programs to keep working. Also, in use cases where there is just one conveyor to track consider leaving this argument out.

#### Example command:

```
track_conveyor_circular(p[0.5,0.5,0,0,0,0], 500.0, false)
```

- Example Parameters:
  - center = p(0.5,0.5,0,0,0,0) → location of the center of the conveyor
  - ticks\_per\_revolution = 500 → the number of ticks the encoder sees when the conveyor moves one revolution
  - rotate\_tool = false → the tool should not rotate with the conveyor, but stay in the orientation specified by the trajectory (move() etc.).

**track\_conveyor\_linear**(*direction*, *ticks\_per\_meter*, *encoder\_index*=0)

Makes robot movement (movej() etc.) track a linear conveyor.

```
>>> track_conveyor_linear(p[1,0,0,0,0,0],1000.0)
```

The example code makes the robot track a conveyor in the x-axis of the robot base coordinate system, where 1000 ticks on the encoder corresponds to 1m along the x-axis.

#### Parameters

- direction:** Pose vector that determines the direction of the conveyor in the base coordinate system of the robot
- ticks\_per\_meter:** How many ticks the encoder sees when the conveyor moves one meter
- encoder\_index:** The index of the encoder to associate with the conveyor tracking. Must be either 0 or 1. This is an optional argument, and please note the default of 0. The ability to omit this argument will allow existing programs to keep working. Also, in use cases where there is just one conveyor to track consider leaving this argument out.

**Example command:** `track_conveyor_linear(p[1,0,0,0,0,0],1000.0)`

- Example Parameters:
  - `direction = p(1,0,0,0,0,0)` → Pose vector that determines the direction of the conveyor in the base coordinate system of the robot
  - `ticks_per_meter = 1000.` → How many ticks the encoder sees when the conveyor moves one meter.

## 2.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'Motion'
<code>a_joint_default</code>	<b>Value:</b> 1.4
<code>a_tool_default</code>	<b>Value:</b> 1.2
<code>v_joint_default</code>	<b>Value:</b> 1.05
<code>v_tool_default</code>	<b>Value:</b> 0.25

## 3 Module internals

### 3.1 Functions

#### **force()**

Returns the force exerted at the TCP

Return the current externally exerted force at the TCP. The force is the norm of Fx, Fy, and Fz calculated using `get_tcp_force()`.

#### **Return Value**

The force in Newtons (float)

**Note:** Refer to `force_mode()` for taring the sensor.

#### **get\_actual\_joint\_positions()**

Returns the actual angular positions of all joints

The angular actual positions are expressed in radians and returned as a vector of length 6. Note that the output might differ from the output of `get_target_joint_positions()`, especially during acceleration and heavy loads.

#### **Return Value**

The current actual joint angular position vector in rad : (Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3)

#### **get\_actual\_joint\_positions\_history(steps=0)**

Returns the actual past angular positions of all joints

This function returns the angular positions as reported by the function "`get_actual_joint_positions()`" which indicates the number of controller time steps occurring before the current time step.

An exception is thrown if indexing goes beyond the buffer size.

#### **Parameters**

`steps`: The number of controller time steps required to go back. 0 corresponds to "`get_actual_joint_positions()`"

#### **Return Value**

The joint angular position vector in rad : (Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3) that was actual at the provided number of steps before the current time step.

**get\_actual\_joint\_speeds()**

Returns the actual angular velocities of all joints

The angular actual velocities are expressed in radians pr. second and returned as a vector of length 6. Note that the output might differ from the output of `get_target_joint_speeds()`, especially during acceleration and heavy loads.

**Return Value**

The current actual joint angular velocity vector in rad/s:  
(Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3)

**get\_actual\_tcp\_pose()**

Returns the current measured tool pose

Returns the 6d pose representing the tool position and orientation specified in the base frame. The calculation of this pose is based on the actual robot encoder readings.

**Return Value**

The current actual TCP vector (X, Y, Z, Rx, Ry, Rz)

**get\_actual\_tcp\_speed()**

Returns the current measured TCP speed

The speed of the TCP returned in a pose structure. The first three values are the cartesian speeds along x,y,z, and the last three define the current rotation axis, rx,ry,rz, and the length |rz,ry,rz| defines the angular velocity in radians/s.

**Return Value**

The current actual TCP velocity vector (X, Y, Z, Rx, Ry, Rz)

**get\_actual\_tool\_flange\_pose()**

Returns the current measured tool flange pose

Returns the 6d pose representing the tool flange position and orientation specified in the base frame, without the Tool Center Point offset. The calculation of this pose is based on the actual robot encoder readings.

**Return Value**

The current actual tool flange vector: (X, Y, Z, Rx, Ry, Rz)

**Note:** See `get_actual_tcp_pose` for the actual 6d pose including TCP offset.

**get\_controller\_temp()**

Returns the temperature of the control box

The temperature of the robot control box in degrees Celcius.

**Return Value**

A temperature in degrees Celcius (float)

**get\_forward\_kin(q='current\_joint\_positions', tcp='active\_tcp')**

Calculate the forward kinematic transformation (joint space -> tool space) using the calibrated robot kinematics. If no joint position vector is provided the current joint angles of the robot arm will be used. If no tcp is provided the currently active tcp of the controller will be used.

**Parameters**

q: joint position vector (Optional)

tcp: tcp offset pose (Optional)

**Return Value**

tool pose

**Example command:** `get_forward_kin([0., 3.14, 1.57, .785, 0, 0], p[0, 0, 0.01, 0, 0, 0])`

- Example Parameters:
  - q = (0., 3.14, 1.57, .785, 0, 0) → joint angles of j0=0 deg, j1=180 deg, j2=90 deg, j3=45 deg, j4=0 deg, j5=0 deg.
  - tcp = p(0, 0, 0.01, 0, 0, 0) → tcp offset of x=0mm, y=0mm, z=10mm and rotation vector of rx=0 deg., ry=0 deg, rz=0 deg.

---

**get\_inverse\_kin**(*x*, *qnear*, *maxPositionError*=1e-10, *maxOrientationError*=1e-10, *tcp*='active\_tcp')

---

Calculate the inverse kinematic transformation (tool space -> joint space). If *qnear* is defined, the solution closest to *qnear* is returned. Otherwise, the solution closest to the current joint positions is returned. If no *tcp* is provided the currently active *tcp* of the controller will be used.

#### Parameters

<i>x</i> :	tool pose
<i>qnear</i> :	list of joint positions (Optional)
<i>maxPositionError</i> :	the maximum allowed position error (Optional)
<i>maxOrientationError</i> :	the maximum allowed orientation error (Optional)
<i>tcp</i> :	tcp offset pose (Optional)

#### Return Value

joint positions

**Example command:** `get_inverse_kin(p[.1,.2,.2,0,3.14,0], [0.,3.14,1.57,.785,0,0])`

- Example Parameters:
  - $x = p(.1,.2,.2,0,3.14,0) \rightarrow$  pose with position of  $x=100\text{mm}$ ,  $y=200\text{mm}$ ,  $z=200\text{mm}$  and rotation vector of  $rx=0\text{ deg.}$ ,  $ry=180\text{ deg.}$ ,  $rz=0\text{ deg.}$
  - $qnear = (0.,3.14,1.57,.785,0,0) \rightarrow$  solution should be near to joint angles of  $j0=0\text{ deg.}$ ,  $j1=180\text{ deg.}$ ,  $j2=90\text{ deg.}$ ,  $j3=45\text{ deg.}$ ,  $j4=0\text{ deg.}$ ,  $j5=0\text{ deg.}$
  - *maxPositionError* is by default  $1e-10\text{ m}$
  - *maxOrientationError* is by default  $1e-10\text{ rad}$

---

#### get\_joint\_temp(*j*)

---

Returns the temperature of joint *j*

The temperature of the joint house of joint *j*, counting from zero.  $j=0$  is the base joint, and  $j=5$  is the last joint before the tool flange.

#### Parameters

*j*: The joint number (int)

#### Return Value

A temperature in degrees Celcius (float)

**get\_joint\_torques()**

Returns the torques of all joints

The torque on the joints, corrected by the torque needed to move the robot itself (gravity, friction, etc.), returned as a vector of length 6.

**Return Value**

The joint torque vector in Nm: (Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3)

**get\_steptime()**

Returns the duration of the robot time step in seconds.

In every time step, the robot controller will receive measured joint positions and velocities from the robot, and send desired joint positions and velocities back to the robot. This happens with a predetermined frequency, in regular intervals. This interval length is the robot time step.

**Return Value**

duration of the robot step in seconds

**get\_target\_joint\_positions()**

Returns the desired angular position of all joints

The angular target positions are expressed in radians and returned as a vector of length 6. Note that the output might differ from the output of `get_actual_joint_positions()`, especially during acceleration and heavy loads.

**Return Value**

The current target joint angular position vector in rad: (Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3)

**get\_target\_joint\_speeds()**

Returns the desired angular velocities of all joints

The angular target velocities are expressed in radians pr. second and returned as a vector of length 6. Note that the output might differ from the output of `get_actual_joint_speeds()`, especially during acceleration and heavy loads.

**Return Value**

The current target joint angular velocity vector in rad/s: (Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3)

**get\_target\_payload()**

Returns the weight of the active payload

**Return Value**

The weight of the current payload in kilograms

**get\_target\_payload\_cog()**

Retrieve the Center Of Gravity (COG) coordinates of the active payload.

This script returns the COG coordinates of the active payload, with respect to the tool flange

**Return Value**

The 3d coordinates of the COG (CoGx, CoGy, CoGz) in meters

**get\_target\_tcp\_pose()**

Returns the current target tool pose

Returns the 6d pose representing the tool position and orientation specified in the base frame. The calculation of this pose is based on the current target joint positions.

**Return Value**

The current target TCP vector (X, Y, Z, Rx, Ry, Rz)

**get\_target\_tcp\_speed()**

Returns the current target TCP speed

The desired speed of the TCP returned in a pose structure. The first three values are the cartesian speeds along x,y,z, and the last three define the current rotation axis, rx,ry,rz, and the length  $|rz,ry,rz|$  defines the angular velocity in radians/s.

**Return Value**

The TCP speed (pose)



**get\_target\_waypoint()**

Returns the target waypoint of the active move

This is different from the `get_target_tcp_pose()` which returns the target pose for each time step. The `get_target_waypoint()` returns the same target pose for `movei`, `movej`, `movep` or `movec` during the motion. It returns the same as `get_target_tcp_pose()`, if none of the mentioned move functions are running.

This method is useful for calculating relative movements where the previous move command uses blends.

**Return Value**

The desired waypoint TCP vector (X, Y, Z, Rx, Ry, Rz)

**get\_tcp\_force()**

Returns the wrench (Force/Torque vector) at the TCP

The function returns `p(Fx(N), Fy(N), Fz(N), TRx(Nm), TRy(Nm), TRz(Nm))` where `Fx`, `Fy`, and `Fz` are the forces in the axes of the robot base coordinate system measured in Newtons, and `TRx`, `TRy`, and `TRz` are the torques around these axes measured in Newton times Meters.

**Return Value**

The wrench (pose)

**Note:** Refer to `force_mode()` for taring the sensor.

**get\_tcp\_offset()**

Gets the active tcp offset, i.e. the transformation from the output flange coordinate system to the TCP as a pose.

**Return Value**

tcp offset pose

**get\_tool\_accelerometer\_reading()**

Returns the current reading of the tool accelerometer as a three-dimensional vector.

The accelerometer axes are aligned with the tool coordinates, and pointing an axis upwards results in a positive reading.

**Return Value**

X, Y, and Z composant of the measured acceleration in SI-units ( $\text{m/s}^2$ ).

**get\_tool\_current()**

Returns the tool current

The tool current consumption measured in ampere.

**Return Value**

The tool current in ampere.

**is\_steady()**

Checks if robot is fully at rest.

True when the robot is fully at rest, and ready to accept higher external forces and torques, such as from industrial screwdrivers. It is useful in combination with the GUI's wait node, before starting the screwdriver or other actuators influencing the position of the robot.

Note: This function will always return false in modes other than the standard position mode, e.g. false in force and teach mode.

**Return Value**

True when the robot is fully at rest. Returns False otherwise (bool)

**is\_within\_safety\_limits(pose)**

Checks if the given pose is reachable and within the current safety limits of the robot.

This check considers joint limits (if the target pose is specified as joint positions), safety planes limits, TCP orientation deviation limits and range of the robot. If a solution is found when applying the inverse kinematics to the given target TCP pose, this pose is considered reachable.

**Parameters**

`pose`: Target pose (which can also be specified as joint positions)

**Return Value**

True if within limits, false otherwise (bool)

**Example command:**

```
is_within_safety_limits(p[.1,.2,.2,0,3.14,0])
```

- Example Parameters:

- `pose = p(.1,.2,.2,0,3.14,0)` → target pose with position of `x=100mm`, `y=200mm`, `z=200mm` and rotation vector of `rx=0 deg.`, `ry=180 deg`, `rz=0 deg`.

**popup(s, title=' Popup' , warning=False, error=False, blocking=False)**

Display popup on GUI

Display message in popup window on GUI.

**Parameters**

- s: message string
- title: title string
- warning: warning message?
- error: error message?
- blocking: if True, program will be suspended until "continue" is pressed

**Example command:** `popup("here I am", title="Popup #1", blocking=True)`

- Example Parameters:
  - s popup text is "here I am"
  - title popup title is "Popup #1"
  - blocking = true → popup must be cleared before other actions will be performed.

**powerdown()**

Shutdown the robot, and power off the robot and controller.

**set\_gravity(d)**

Set the direction of the acceleration experienced by the robot. When the robot mounting is fixed, this corresponds to an acceleration of g away from the earth's centre.

```
>>> set_gravity([0, 9.82*sin(theta), 9.82*cos(theta)])
```

will set the acceleration for a robot that is rotated "theta" radians around the x-axis of the robot base coordinate system

**Parameters**

- d: 3D vector, describing the direction of the gravity, relative to the base of the robot.

**Example command:** `set_gravity(0, 9.82, 0)`

- Example Parameters:
  - d is vector with a direction of y (direction of the robot cable) and a magnitude of 9.82 m/s<sup>2</sup> (1g).

**set\_payload(m, cog)**

Set payload mass and center of gravity

Sets the mass and center of gravity (abbr. CoG) of the payload.

This function must be called, when the payload weight or weight distribution changes - i.e when the robot picks up or puts down a heavy workpiece.

**Parameters**

**m:** mass in kilograms

**cog:** Center of Gravity, a vector (CoGx, CoGy, CoGz) specifying the displacement (in meters) from the toolmount.

**Warning:** Omitting the cog parameter is **deprecated**. The Tool Center Point (TCP) will be used if the cog parameter is missing with the side effect that later calls to `set_tcp` will change also the CoG to the new TCP. Use the `set_payload_mass` function to change only the mass or use the `get_target_payload_cog` as second argument to not change the CoG.

**Example command:**

- `set_payload(3., [0,0,.3])`
  - Example Parameters:
    - \* `m = 3` → mass is set to 3 kg payload
    - \* `cog = (0,0,.3)` → Center of Gravity is set to x=0 mm, y=0 mm, z=300 mm from the center of the tool mount in tool coordinates
- `set_payload(2.5, get_target_payload_cog())`
  - Example Parameters:
    - \* `m = 2.5` → mass is set to 2.5 kg payload
    - \* `cog` = use the current COG setting

**set\_payload\_cog(CoG)**

Set the Center of Gravity (CoG)

**Deprecated:** This function is deprecated. It is recommended to set **always** the CoG with the mass (see `set_payload`).

**set\_payload\_mass(m)**

Set payload mass

See also `set_payload`.

Sets the mass of the payload and leaves the center of gravity (CoG) unchanged.

**Parameters**

`m`: mass in kilograms

**Example command:** `set_payload_mass(3.)`

- Example Parameters:
  - `m = 3` → mass is set to 3 kg payload

**set\_tcp(pose)**

Sets the active tcp offset, i.e. the transformation from the output flange coordinate system to the TCP as a pose.

**Parameters**

`pose`: A pose describing the transformation.

**Example command:** `set_tcp(p[0., .2, .3, 0., 3.14, 0.])`

- Example Parameters:
  - `pose = p(0.,.2,.3,0.,3.14,0.)` → tool center point is set to `x=0mm, y=200mm, z=300mm`, rotation vector is `rx=0 deg, ry=180 deg, rz=0 deg`. In tool coordinates

**sleep(t)**

Sleep for an amount of time

**Parameters**

`t`: time (s)

**Example command:** `sleep(3.)`

- Example Parameters:
  - `t = 3.` → time to sleep

**str\_at(src, index)**

Provides direct access to the bytes of a string.

This script returns a string containing the byte in the source string at the position corresponding to the specified `index`. It may not correspond to an actual character in case of strings with special encoded character (i.e. multi-byte or variable-length encoding)

The string is zero-indexed.

**Parameters**

`src`: source string.

`index`: integer specifying the position inside the source string.

**Return Value**

String containing the byte at position `index` in the source string. An exception is raised if the index is not valid.

**Example command:**

- `str_at("Hello", 0)`
  - returns "H"
- `str_at("Hello", 1)`
  - returns "e"
- `str_at("Hello", 10)`
  - error (index out of bound)
- `str_at("", 0)`
  - error (source string is empty)

**str\_cat(op1, op2)**

## String concatenation

This script returns a string that is the concatenation of the two operands given as input. Both operands can be one of the following types: String, Boolean, Integer, Float, Pose, List of Boolean / Integer / Float / Pose. Any other type will raise an exception.

The resulting string cannot exceed 1023 characters, an exception is thrown otherwise.

Float numbers will be formatted with 6 decimals, and trailing zeros will be removed.

The function can be nested to create complex strings (see last example).

**Parameters**

op1: first operand

op2: second operand

**Return Value**

String concatenation of op1 and op2

**Example command:**

- `str_cat("Hello", " World!")`
  - returns "Hello World!"
- `str_cat("Integer ", 1)`
  - returns "Integer 1"
- `str_cat("", p[1.0, 2.0, 3.0, 4.0, 5.0, 6.0])`
  - returns "p(1,2,3,4,5,6)"
- `str_cat([True, False, True], [1, 0, 1])`
  - returns "(True, False, True)(1, 0, 1)"
- `str_cat(str_cat("", str_cat("One", "Two")), str_cat(3, 4))`
  - returns "OneTwo34"

**str\_empty(str)**

Returns true when `str` is empty, false otherwise.

**Parameters**

`str`: source string.

**Return Value**

True if the string is empty, false otherwise

**Example command:**

- `str_empty("")`
  - returns True
- `str_empty("Hello")`
  - returns False

**str\_find(src, target, start\_from=0)**

Finds the first occurrence of the substring `target` in `src`.

This script returns the index (i.e. byte) of the the first occurrence of substring `target` in `str`, starting from the given (optional) position.

The result may not correspond to the actual position of the first character of `target` in case `src` contains multi-byte or variable-length encoded characters.

The string is zero-indexed.

**Parameters**

`src`: source string.

`target`: substring to search.

`start_from`: optional starting position (default 0).

**Return Value**

The index of the first occurrence of `target` in `src`, -1 if `target` is not found in `src`.

**Example command:**

- `str_find("Hello World!", "o")`
  - returns 4
- `str_find("Hello World!", "lo")`
  - returns 3
- `str_find("Hello World!", "o", 5)`
  - returns 7
- `str_find("abc", "z")`
  - returns -1



**str\_len(str)**

---

Returns the number of bytes in a string.

Please note that the value returned may not correspond to the actual number of characters in sequences of multi-byte or variable-length encoded characters.

The string is zero-indexed.

**Parameters**

`str`: source string.

**Return Value**

The number of bytes in the input string.

**Example command:**

- `str_len("Hello")`
  - returns 5
- `str_len("")`
  - returns 0

**str\_sub(src, index, len)**

Returns a substring of `src`.

The result is the substring of `src` that starts at the byte specified by `index` with length of at most `len` bytes. If the requested substring extends past the end of the original string (i.e. `index + len > src length`), the length of the resulting substring is limited to the size of `src`.

An exception is thrown in case `index` and/or `len` are out of bounds. The string is zero-indexed.

**Parameters**

- `src`: source string.
- `index`: integer value specifying the initial byte in the range (0, `src length`)
- `len`: (optional) length of the substring in the range (0, `MAX_INT`). If `len` is not specified, the string in the range (`index`, `src length`).

**Return Value**

the portion of `src` that starts at byte `index` and spans `len` characters.

**Example command:**

- `str_sub("0123456789abcdefghij", 5, 3)`
  - returns "567"
- `str_sub("0123456789abcdefghij", 10)`
  - returns "abcdefghij"
- `str_sub("0123456789abcdefghij", 2, 0)`
  - returns "" (len is 0)
- `str_sub("abcde", 2, 50)`
  - returns "cde"
- `str_sub("abcde", -5, 50)`
  - error: index is out of bounds

**sync()**

Uses up the remaining "physical" time a thread has in the current frame.

**textmsg**(s1, s2='')

---

Send text message to log

Send message with s1 and s2 concatenated to be shown on the GUI log-tab

**Parameters**

s1: message string, variables of other types (int, bool poses etc.) can also be sent

s2: message string, variables of other types (int, bool poses etc.) can also be sent

**Example command:** `textmsg("value=", 3)`

- Example Parameters:
  - s1 set first part of message to "value="
  - s2 set second part of message to 3
    - \* message in the log is "value=3"

**to\_num(str)**

Converts a string to a number.

`to_num` returns an integer or a float depending on the presence of a decimal point in the input string. Only `'.'` is recognized as decimal point independent of locale settings.

Valid strings can contains optional leading white space(s) followed by an optional plus (`'+'`) or minus sign (`'-'`) and then one of the following:

(i) A decimal number consisting of a sequence of decimal digits (e.g. 10, -5), an optional `'.'` to indicate a float number (e.g. 1.5234, -2.0, .36) and a optional decimal exponent that indicates multiplication by a power of 10 (e.g. 10e3, 2.5E-5, -5e-4).

(ii) A hexadecimal number consisting of `"0x"` or `"0X"` followed by a nonempty sequence of hexadecimal digits (e.g. `"0X3A"`, `"0xb5"`).

(iii) An infinity (either `"INF"` or `"INFINITY"`, case insensitive)

(iv) A Not-a-Number (`"NAN"`, case insensitive)

Runtime exceptions are raised if the source string doesn't contain a valid number or the result is out of range for the resulting type.

**Parameters**

`str`: string to convert

**Return Value**

Integer or float number according to the input string.

**Example command:**

- `to_num("10")`
  - returns 10 //integer
- `to_num("3.14")`
  - returns 3.14 //float
- `to_num("-3.0e5")`
  - returns -3.0e5 //float due to `'.'` in the input string
- `to_num("+5.")`
  - returns 5.0 //float due to `'.'` in the input string
- `to_num("123abc")`
  - error string doesn't contain a valid number

**to\_str(val)**

Gets string representation of a value.

This script converts a value of type Boolean, Integer, Float, Pose (or a list of those types) to a string.

The resulting string cannot exceed 1023 characters.

Float numbers will be formatted with 6 decimals, and trailing zeros will be removed.

**Parameters**

`val`: value to convert

**Return Value**

The string representation of the given value.

**Example command:**

- `to_str(10)`
  - returns "10"
- `to_str(2.123456123456)`
  - returns "2.123456"
- `to_str(p[1.0, 2.0, 3.0, 4.0, 5.0, 6.0])`
  - returns "p(1, 2, 3, 4, 5, 6)"
- `to_str([True, False, True])`
  - returns "(True, False, True)"

**tool\_contact(direction)**

Detects when a contact between the tool and an object happens.

**Parameters**

`direction`: List of six floats. The first three elements are interpreted as a 3D vector (in the robot base coordinate system) giving the direction in which contacts should be detected. If all elements of the list are zero, contacts from all directions are considered.

**Return Value**

Integer. The returned value is the number of time steps back to just before the contact have started. A value larger than 0 means that a contact is detected. A value of 0 means no contact.

**tool\_contact\_examples()**

Example of usage in conjunction with the "get\_actual\_joint\_positions\_history()" function to allow the robot to retract to the initial point of contact:

```
>>> def testToolContact():
>>>     while True:
>>>         step_back = tool_contact()
>>>         if step_back <= 0:
>>>             # Continue moving with 100mm/s
>>>             speedl([0,0,-0.100,0,0,0], 0.5, t=get_steptime())
>>>         else:
>>>             # Contact detected!
>>>             # Get q for when the contact was first seen
>>>             q = get_actual_joint_positions_history(step_back)
>>>             # Stop the movement
>>>             stopl(3)
>>>             # Move to the initial contact point
>>>             movel(q)
>>>             break
>>>         end
>>>     end
>>> end
```

**Example command:** tool\_contact(direction = get\_target\_tcp\_speed())

- Example Parameters:
  - direction=get\_target\_tcp\_speed() will detect contacts in the direction of TCP movement

tool\_contact(direction = [1,0,0,0,0,0])

- Example Parameters:
  - direction=(1,0,0,0,0,0) will detect contacts in the direction robot base X

## 3.2 Variables

Name	Description
__package__	<b>Value:</b> None

## 4 Module urmath

### 4.1 Functions

**acos(f)**

Returns the arc cosine of f

Returns the principal value of the arc cosine of f, expressed in radians. A runtime error is raised if f lies outside the range (-1, 1).

**Parameters**

f: floating point value

**Return Value**

the arc cosine of f.

**Example command:** `acos(0.707)`

- Example Parameters:
  - f is the cos of 45 deg. (.785 rad)
  - \* Returns .785

**asin(f)**

Returns the arc sine of f

Returns the principal value of the arc sine of f, expressed in radians. A runtime error is raised if f lies outside the range (-1, 1).

**Parameters**

f: floating point value

**Return Value**

the arc sine of f.

**Example command:** `asin(0.707)`

- Example Parameters:
  - f is the sin of 45 deg. (.785 rad)
  - \* Returns .785

**atan(f)**

Returns the arc tangent of f

Returns the principal value of the arc tangent of f, expressed in radians.

**Parameters**

f: floating point value

**Return Value**

the arc tangent of f.

**Example command:** `atan(1.)`

- Example Parameters:
  - f is the tan of 45 deg. (.785 rad)
  - \* Returns .785

**atan2(x, y)**

Returns the arc tangent of x/y

Returns the principal value of the arc tangent of x/y, expressed in radians. To compute the value, the function uses the sign of both arguments to determine the quadrant.

**Parameters**

x: floating point value

y: floating point value

**Return Value**

the arc tangent of x/y.

**Example command:** `atan2(.5, .5)`

- Example Parameters:
  - x is the one side of the triangle
  - y is the second side of a triangle
  - \* Returns  $\text{atan}(.5/.5) = .785$



**binary\_list\_to\_integer(l)**

Returns the value represented by the content of list l

Returns the integer value represented by the bools contained in the list l when evaluated as a signed binary number.

**Parameters**

- l: The list of bools to be converted to an integer. The bool at index 0 is evaluated as the least significant bit. False represents a zero and True represents a one. If the list is empty this function returns 0. If the list contains more than 32 bools, the function returns the signed integer value of the first 32 bools in the list.

**Return Value**

The integer value of the binary list content.

**Example command:**

```
binary_list_to_integer([True, False, False, True])
```

- Example Parameters:
  - l represents the binary values 1001
  - \* Returns 9

**ceil(f)**

Returns the smallest integer value that is not less than f

Rounds floating point number to the smallest integer no greater than f.

**Parameters**

- f: floating point value

**Return Value**

rounded integer

**Example command:** `ceil(1.43)`

- Example Parameters:
  - Returns 2

**cos(*f*)**

Returns the cosine of *f*

Returns the cosine of an angle of *f* radians.

**Parameters**

*f*: floating point value

**Return Value**

the cosine of *f*.

**Example command:** `cos(1.57)`

- Example Parameters:
  - *f* is angle of 1.57 rad (90 deg)
  - \* Returns 0.0

**d2r(*d*)**

Returns degrees-to-radians of *d*

Returns the radian value of '*d*' degrees. Actually:  $(d/180)*\text{MATH\_PI}$

**Parameters**

*d*: The angle in degrees

**Return Value**

The angle in radians

**Example command:** `d2r(90)`

- Example Parameters:
  - *d* angle in degrees
  - \* Returns 1.57 angle in radians

**floor(*f*)**

Returns largest integer not greater than *f*

Rounds floating point number to the largest integer no greater than *f*.

**Parameters**

*f*: floating point value

**Return Value**

rounded integer

**Example command:** `floor(1.53)`

- Example Parameters:
  - Returns 1

**get\_list\_length(v)**

Returns the length of a list variable

The length of a list is the number of entries the list is composed of.

**Parameters**

v: A list variable

**Return Value**

An integer specifying the length of the given list

**Example command:** `get_list_length([1, 3, 3, 6, 2])`

- Example Parameters:
  - v is the list 1,3,3,6,2
  - \* Returns 5

**integer\_to\_binary\_list(x)**

Returns the binary representation of x

Returns a list of bools as the binary representation of the signed integer value x.

**Parameters**

x: The integer value to be converted to a binary list.

**Return Value**

A list of 32 bools, where False represents a zero and True represents a one. The bool at index 0 is the least significant bit.

**Example command:** `integer_to_binary_list(57)`

- Example Parameters:
  - x integer 57
  - \* Returns binary list

**interpolate\_pose(*p\_from*, *p\_to*, *alpha*)**

Linear interpolation of tool position and orientation.

When alpha is 0, returns *p\_from*. When alpha is 1, returns *p\_to*. As alpha goes from 0 to 1, returns a pose going in a straight line (and geodetic orientation change) from *p\_from* to *p\_to*. If alpha is less than 0, returns a point before *p\_from* on the line. If alpha is greater than 1, returns a pose after *p\_to* on the line.

**Parameters**

- p\_from*: tool pose (pose)
- p\_to*: tool pose (pose)
- alpha*: Floating point number

**Return Value**

interpolated pose (pose)

**Example command:** `interpolate_pose(p[.2, .2, .4, 0, 0, 0], p[.2, .2, .6, 0, 0, 0], .5)`

- Example Parameters:
  - *p\_from* = `p(.2,.2,.4,0,0,0)`
  - *p\_to* = `p(.2,.2,.6,0,0,0)`
  - *alpha* = `.5`
  - \* Returns `p(.2,.2,.5,0,0,0)`

**length(*v*)**

Returns the length of a list variable or a string

The length of a list or string is the number of entries or characters it is composed of.

**Parameters**

- v*: A list or string variable

**Return Value**

An integer specifying the length of the given list or string

**Example command:** `length("here I am")`

- Example Parameters:
  - *v* equals string "here I am"
  - \* Returns 9

**log(b, f)**

Returns the logarithm of f to the base b

Returns the logarithm of f to the base b. If b or f is negative, or if b is 1 a runtime error is raised.

**Parameters**

b: floating point value

f: floating point value

**Return Value**

the logarithm of f to the base of b.

**Example command:** `log(10., 4.)`

- Example Parameters:
  - b is base 10
  - f is log of 4
  - \* Returns 0.60206

**norm(a)**

Returns the norm of the argument

The argument can be one of four different types:

Pose: In this case the euclidian norm of the pose is returned.

Float: In this case fabs(a) is returned.

Int: In this case abs(a) is returned.

List: In this case the euclidian norm of the list is returned, the list elements must be numbers.

**Parameters**

a: Pose, float, int or List

**Return Value**

norm of a

**Example command:**

- `norm(-5.3)` → Returns 5.3
- `norm(-8)` → Returns 8
- `norm(p[-.2, .2, -.2, -1.57, 0, 3.14])` → Returns 3.52768

**normalize(v)**

Returns the normalized form of a list of floats

Except for the case of all zeroes, the normalized form corresponds to the unit vector in the direction of v.

Throws an exception if the sum of all squared elements is zero.

**Parameters**

v: List of floats

**Return Value**

normalized form of v

**Example command:**

- `normalize([1, 0, 0])` → Returns (1, 0, 0)
- `normalize([0, 5, 0])` → Returns (0, 1, 0)
- `normalize([0, 1, 1])` → Returns (0, 0.707, 0.707)

**point\_dist(p\_from, p\_to)**

Point distance

**Parameters**

p\_from: tool pose (pose)

p\_to: tool pose (pose)

**Return Value**

Distance between the two tool positions (without considering rotations)

**Example command:** `point_dist(p[.2, .5, .1, 1.57, 0, 3.14], p[.2, .5, .6, 0, 1.57, 3.14])`

- Example Parameters:
  - `p_from = p(.2,.5,.1,1.57,0,3.14)` → The first point
  - `p_to = p(.2,.5,.6,0,1.57,3.14)` → The second point
  - \* Returns distance between the points regardless of rotation

**pose\_add(p\_1, p\_2)**

Pose addition

Both arguments contain three position parameters (x, y, z) jointly called P, and three rotation parameters (R\_x, R\_y, R\_z) jointly called R. This function calculates the result x\_3 as the addition of the given poses as follows:

$$p_3.P = p_1.P + p_2.P$$

$$p_3.R = p_1.R * p_2.R$$

**Parameters**

p\_1: tool pose 1 (pose)

p\_2: tool pose 2 (pose)

**Return Value**

Sum of position parts and product of rotation parts (pose)

**Example command:** `pose_add(p[.2, .5, .1, 1.57, 0, 0],  
p[.2, .5, .6, 1.57, 0, 0])`

- Example Parameters:
  - p\_1 = p(.2,.5,.1,1.57,0,0) → The first point
  - p\_2 = p(.2,.5,.6,1.57,0,0) → The second point
  - \* Returns p(0.4,1.0,0.7,3.14,0,0)

**pose\_dist(p\_from, p\_to)**

Pose distance

**Parameters**

p\_from: tool pose (pose)

p\_to: tool pose (pose)

**Return Value**

distance

**Example command:** `pose_dist(p[.2, .5, .1, 1.57, 0, 3.14],  
p[.2, .5, .6, 0, 1.57, 3.14])`

- Example Parameters:
  - p\_from = p(.2,.5,.1,1.57,0,3.14) → The first point
  - p\_to = p(.2,.5,.6,0,1.57,3.14) → The second point
  - \* Returns distance between two poses including rotation

**pose\_inv(p\_from)**

Get the inverse of a pose

**Parameters**

p\_from: tool pose (spatial vector)

**Return Value**

inverse tool pose transformation (spatial vector)

**Example command:** `pose_inv(p[.2, .5, .1, 1.57, 0, 3.14])`

- Example Parameters:
  - p\_from = p(.2,.5,.1,1.57,0,3.14) → The point
  - \* Returns p(0.19324,0.41794,-0.29662,1.23993,0.0,2.47985)

**pose\_sub(p\_to, p\_from)**

Pose subtraction

**Parameters**

p\_to: tool pose (spatial vector)

p\_from: tool pose (spatial vector)

**Return Value**

tool pose transformation (spatial vector)

**Example command:** `pose_sub(p[.2, .5, .1, 1.57, 0, 0],  
p[.2, .5, .6, 1.57, 0, 0])`

- Example Parameters:
  - p\_1 = p(.2,.5,.1,1.57,0,0) → The first point
  - p\_2 = p(.2,.5,.6,1.57,0,0) → The second point
  - \* Returns p(0.0,0.0,-0.5,0.0,.0,.0)



**pose\_trans**(p\_from, p\_from\_to)

Pose transformation

The first argument, p\_from, is used to transform the second argument, p\_from\_to, and the result is then returned. This means that the result is the resulting pose, when starting at the coordinate system of p\_from, and then in that coordinate system moving p\_from\_to.

This function can be seen in two different views. Either the function transforms, that is translates and rotates, p\_from\_to by the parameters of p\_from. Or the function is used to get the resulting pose, when first making a move of p\_from and then from there, a move of p\_from\_to.

If the poses were regarded as transformation matrices, it would look like:

$$T_{\text{world} \rightarrow \text{to}} = T_{\text{world} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}}$$

$$T_{\text{x} \rightarrow \text{to}} = T_{\text{x} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}}$$

**Parameters**

- p\_from: starting pose (spatial vector)
- p\_from\_to: pose change relative to starting pose (spatial vector)

**Return Value**

resulting pose (spatial vector)

**Example command:** pose\_trans(p[.2, .5, .1, 1.57, 0, 0],  
p[.2, .5, .6, 1.57, 0, 0])

- Example Parameters:
  - p\_1 = p(.2,.5,.1,1.57,0,0) → The first point
  - p\_2 = p(.2,.5,.6,1.57,0,0) → The second point
  - \* Returns p(0.4,-0.0996,0.60048,3.14,0.0,0.0)

**pow(base, exponent)**

Returns base raised to the power of exponent

Returns the result of raising base to the power of exponent. If base is negative and exponent is not an integral value, or if base is zero and exponent is negative, a runtime error is raised.

**Parameters**

base: floating point value

exponent: floating point value

**Return Value**

base raised to the power of exponent

**Example command:** `pow(5., 3)`

- Example Parameters:
  - Base = 5
  - Exponent = 3
  - \* Returns 125.

**r2d(r)**

Returns radians-to-degrees of r

Returns the degree value of 'r' radians.

**Parameters**

r: The angle in radians

**Return Value**

The angle in degrees

**Example command:** `r2d(1.57)`

- Example Parameters:
  - r 1.5707 rad
  - \* Returns 90 deg

**random()**

Random Number

**Return Value**

pseudo-random number between 0 and 1 (float)

**rotvec2rpy(rotation\_vector)**

Returns RPY vector corresponding to rotation\_vector

Returns the RPY vector corresponding to 'rotation\_vector' where the rotation vector is the axis of rotation with a length corresponding to the angle of rotation in radians.

**Parameters**

`rotation_vector`: The rotation vector (Vector3d) in radians, also called the Axis-Angle vector (unit-axis of rotation multiplied by the rotation angle in radians).

**Return Value**

The RPY vector (Vector3d) in radians, describing a roll-pitch-yaw sequence of extrinsic rotations about the X-Y-Z axes, (corresponding to intrinsic rotations about the Z-Y'-X'' axes). In matrix form the RPY vector is defined as  $R_{rpy} = R_z(\text{yaw})R_y(\text{pitch})R_x(\text{roll})$ .

**Example command:** `rotvec2rpy([3.14, 1.57, 0])`

- Example Parameters:
  - `rotation_vector = (3.14, 1.57, 0)` → `rx=3.14, ry=1.57, rz=0`
  - \* Returns `(-2.80856, -0.16202, 0.9)` → `roll=-2.80856, pitch=-0.16202, yaw=0.9`

**rpy2rotvec(rpy\_vector)**

Returns rotation vector corresponding to rpy\_vector

Returns the rotation vector corresponding to 'rpy\_vector' where the RPY (roll-pitch-yaw) rotations are extrinsic rotations about the X-Y-Z axes (corresponding to intrinsic rotations about the Z-Y'-X'' axes).

**Parameters**

**rpy\_vector**: The RPY vector (Vector3d) in radians, describing a roll-pitch-yaw sequence of extrinsic rotations about the X-Y-Z axes, (corresponding to intrinsic rotations about the Z-Y'-X'' axes). In matrix form the RPY vector is defined as  $R_{rpy} = R_z(yaw)R_y(pitch)R_x(roll)$ .

**Return Value**

The rotation vector (Vector3d) in radians, also called the Axis-Angle vector (unit-axis of rotation multiplied by the rotation angle in radians).

**Example command:** `rpy2rotvec([3.14, 1.57, 0])`

- Example Parameters:
  - `rpy_vector = (3.14, 1.57, 0)` → roll=3.14, pitch=1.57, yaw=0
  - \* Returns (2.22153, 0.00177, -2.21976) → rx=2.22153, ry=0.00177, rz=-2.21976

**sin(f)**

Returns the sine of f

Returns the sine of an angle of f radians.

**Parameters**

**f**: floating point value

**Return Value**

the sine of f.

**Example command:** `sin(1.57)`

- Example Parameters:
  - f is angle of 1.57 rad (90 deg)
  - \* Returns 1.0

**sqrt(f)**

Returns the square root of f

Returns the square root of f. If f is negative, a runtime error is raised.

**Parameters**

f: floating point value

**Return Value**

the square root of f.

**Example command:** `sqrt(9)`

- Example Parameters:
  - f = 9
  - \* Returns 3

**tan(f)**

Returns the tangent of f

Returns the tangent of an angle of f radians.

**Parameters**

f: floating point value

**Return Value**

the tangent of f.

**Example command:** `tan(.7854)`

- Example Parameters:
  - f is angle of .7854 rad (45 deg)
  - \* Returns 1.0

**wrench\_trans**( $T_{from\_to}$ ,  $w_{from}$ )

Wrench transformation

Move the point of view of a wrench.

Note: Transforming wrenches is not as trivial as transforming poses as the torque scales with the length of the translation.

$w_{to} = T_{from \rightarrow to} * w_{from}$

**Parameters**

$T_{from\_to}$ : The transformation to the new point of view (Pose)

$w_{from}$ : wrench to transform in list format ( $F_x$ ,  $F_y$ ,  $F_z$ ,  $M_x$ ,  $M_y$ ,  $M_z$ )

**Return Value**

resulting wrench,  $w_{to}$  in list format ( $F_x$ ,  $F_y$ ,  $F_z$ ,  $M_x$ ,  $M_y$ ,  $M_z$ )

## 4.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> None

## 5 Module interfaces

### 5.1 Functions

#### **get\_analog\_in(n)**

*Deprecated:* Get analog input signal level

##### **Parameters**

n: The number (id) of the input, integer: (0:3)

##### **Return Value**

float, The signal level in Amperes, or Volts

**Deprecated:** The `get_standard_analog_in` and `get_tool_analog_in` replace this function. Ports 2-3 should be changed to 0-1 for the latter function. This function might be removed in the next major release.

**Note:** For backwards compatibility n:2-3 go to the tool analog inputs.

**Example command:** `get_analog_in(1)`

- Example Parameters:
  - n is analog input 1
  - \* Returns value of analog output #1

#### **get\_analog\_out(n)**

*Deprecated:* Get analog output signal level

##### **Parameters**

n: The number (id) of the output, integer: (0:1)

##### **Return Value**

float, The signal level in Amperes, or Volts

**Deprecated:** The `get_standard_analog_out` replaces this function. This function might be removed in the next major release.

**Example command:** `get_analog_out(1)`

- Example Parameters:
  - n is analog output 1
  - \* Returns value of analog output #1

**get\_configurable\_digital\_in(*n*)**

Get configurable digital input signal level

See also `get_standard_digital_in` and `get_tool_digital_in`.

**Parameters**

*n*: The number (id) of the input, integer: (0:7)

**Return Value**

boolean, The signal level.

**Example command:** `get_configurable_digital_in(1)`

- Example Parameters:
  - *n* is configurable digital input 1
  - \* Returns True or False

**get\_configurable\_digital\_out(*n*)**

Get configurable digital output signal level

See also `get_standard_digital_out` and `get_tool_digital_out`.

**Parameters**

*n*: The number (id) of the output, integer: (0:7)

**Return Value**

boolean, The signal level.

**Example command:** `get_configurable_digital_out(1)`

- Example Parameters:
  - *n* is configurable digital output 1
  - \* Returns True or False



**get\_digital\_in(n)**

*Deprecated:* Get digital input signal level

**Parameters**

n: The number (id) of the input, integer: (0:9)

**Return Value**

boolean, The signal level.

**Deprecated:** The `get_standard_digital_in` and `get_tool_digital_in` replace this function. Ports 8-9 should be changed to 0-1 for the latter function. This function might be removed in the next major release.

**Note:** For backwards compatibility n:8-9 go to the tool digital inputs.

**Example command:** `get_digital_in(1)`

- Example Parameters:
  - n is digital input 1
  - \* Returns True or False

**get\_digital\_out(n)**

*Deprecated:* Get digital output signal level

**Parameters**

n: The number (id) of the output, integer: (0:9)

**Return Value**

boolean, The signal level.

**Deprecated:** The `get_standard_digital_out` and `get_tool_digital_out` replace this function. Ports 8-9 should be changed to 0-1 for the latter function. This function might be removed in the next major release.

**Note:** For backwards compatibility n:8-9 go to the tool digital outputs.

**Example command:** `get_digital_out(1)`

- Example Parameters:
  - n is digital output 1
  - \* Returns True or False

**get\_flag(n)**

Flags behave like internal digital outputs. They keep information between program runs.

**Parameters**

n: The number (id) of the flag, integer: (0:31)

**Return Value**

Boolean, The stored bit.

**Example command:** `get_flag(1)`

- Example Parameters:
  - n is flag number 1
  - \* Returns True or False

**get\_standard\_analog\_in(n)**

Get standard analog input signal level

See also `get_tool_analog_in`.

**Parameters**

n: The number (id) of the input, integer: (0:1)

**Return Value**

float, The signal level in Amperes, or Volts

**Example command:** `get_standard_analog_in(1)`

- Example Parameters:
  - n is standard analog input 1
  - \* Returns value of standard analog input #1

**get\_standard\_analog\_out(n)**

Get standard analog output signal level

**Parameters**

n: The number (id) of the output, integer: (0:1)

**Return Value**

float, The signal level in Amperes, or Volts

**Example command:** `get_standard_analog_out(1)`

- Example Parameters:
  - n is standard analog output 1
  - \* Returns value of standard analog output #1

**get\_standard\_digital\_in(n)**

Get standard digital input signal level

See also `get_configurable_digital_in` and `get_tool_digital_in`.

**Parameters**

n: The number (id) of the input, integer: (0:7)

**Return Value**

boolean, The signal level.

**Example command:** `get_standard_digital_in(1)`

- Example Parameters:
  - n is standard digital input 1
  - \* Returns True or False

**get\_standard\_digital\_out(n)**

Get standard digital output signal level

See also `get_configurable_digital_out` and `get_tool_digital_out`.

**Parameters**

n: The number (id) of the output, integer: (0:7)

**Return Value**

boolean, The signal level.

**Example command:** `get_standard_digital_out(1)`

- Example Parameters:
  - n is standard digital output 1
  - \* Returns True or False

**get\_tool\_analog\_in(*n*)**

Get tool analog input signal level

See also `get_standard_analog_in`.

**Parameters**

*n*: The number (id) of the input, integer: (0:1)

**Return Value**

float, The signal level in Amperes, or Volts

**Example command:** `get_tool_analog_in(1)`

- Example Parameters:
  - *n* is tool analog input 1
  - \* Returns value of tool analog input #1

**get\_tool\_digital\_in(*n*)**

Get tool digital input signal level

See also `get_configurable_digital_in` and `get_standard_digital_in`.

**Parameters**

*n*: The number (id) of the input, integer: (0:1)

**Return Value**

boolean, The signal level.

**Example command:** `get_tool_digital_in(1)`

- Example Parameters:
  - *n* is tool digital input 1
  - \* Returns True or False

**get\_tool\_digital\_out(n)**

Get tool digital output signal level

See also `get_standard_digital_out` and `get_configurable_digital_out`.

**Parameters**

n: The number (id) of the output, integer: (0:1)

**Return Value**

boolean, The signal level.

**Example command:** `get_tool_digital_out(1)`

- Example Parameters:
  - n is tool digital out 1
  - \* Returns True or False

**get\_tool\_digital\_output\_mode(n)**

Get the output mode of tool output pin.

**Parameters**

n: The number (id) of the output, Integer: (0:1)

**Return Value**

Integer (1:3), pin output mode. 1 = Sinking/NPN, 2 = Sourcing/PNP, 3 = Push-Pull

**Example command:** `get_tool_digital_output_mode(1)`

- n is tool digital output 1.
  - Returns Integer (1:3)

**get\_tool\_output\_mode()**

Get tool Digital Outputs mode.

**Return Value**

Integer (0:1): 0 = Digital Output Mode, 1 = Power(dual pin) Mode

**modbus\_add\_signal**(*IP, slave\_number, signal\_address, signal\_type, signal\_name, sequential\_mode=False*)

Adds a new modbus signal for the controller to supervise. Expects no response.

```
>>> modbus_add_signal("172.140.17.11", 255, 5, 1, "output1")
```

#### Parameters

IP:	A string specifying the IP address of the modbus unit to which the modbus signal is connected.
slave_number:	An integer normally not used and set to 255, but is a free choice between 0 and 255.
signal_address:	An integer specifying the address of the either the coil or the register that this new signal should reflect. Consult the configuration of the modbus unit for this information.
signal_type:	An integer specifying the type of signal to add. 0 = digital input, 1 = digital output, 2 = register input and 3 = register output.
signal_name:	A string uniquely identifying the signal. If a string is supplied which is equal to an already added signal, the new signal will replace the old one. The length of the string cannot exceed 20 characters.
sequential_mode:	Setting to True forces the modbus client to wait for a response before sending the next request. This mode is required by some fieldbus units (Optional).

**Example command:** `modbus_add_signal("172.140.17.11", 255, 5, 1, "output1")`

- Example Parameters:
  - IP address = 172.140.17.11
  - Slave number = 255
  - Signal address = 5
  - Signal type = 1 digital output
  - Signal name = output 1

**modbus\_delete\_signal**(*signal\_name*)

Deletes the signal identified by the supplied signal name.

```
>>> modbus_delete_signal("output1")
```

**Parameters**

*signal\_name*: A string equal to the name of the signal that should be deleted.

**Example command:** `modbus_delete_signal("output1")`

- Example Parameters:
  - Signal name = output1

**modbus\_get\_signal\_status**(*signal\_name*, *is\_secondary\_program*)

Reads the current value of a specific signal.

```
>>> modbus_get_signal_status("output1", False)
```

**Parameters**

*signal\_name*: A string equal to the name of the signal for which the value should be gotten.

*is\_secondary\_program*: A boolean for internal use only. Must be set to False.

**Return Value**

An integer or a boolean. For digital signals: True or False. For register signals: The register value expressed as an unsigned integer.

**Example command:**

```
modbus_get_signal_status("output1", False)
```

- Example Parameters:
  - Signal name = output 1
  - Is\_secondary\_program = False (Note: must be set to False)

**modbus\_send\_custom\_command**(*IP, slave\_number, function\_code, data*)

Sends a command specified by the user to the modbus unit located on the specified IP address. Cannot be used to request data, since the response will not be received. The user is responsible for supplying data which is meaningful to the supplied function code. The builtin function takes care of constructing the modbus frame, so the user should not be concerned with the length of the command.

```
>>> modbus_send_custom_command("172.140.17.11", 103, 6,
>>>                               [17, 32, 2, 88])
```

The above example sets the watchdog timeout on a Beckhoff BK9050 to 600 ms. That is done using the modbus function code 6 (preset single register) and then supplying the register address in the first two bytes of the data array ((17,32) = (0x1120)) and the desired register content in the last two bytes ((2,88) = (0x0258) = dec 600).

**Parameters**

<b>IP:</b>	A string specifying the IP address locating the modbus unit to which the custom command should be send.
<b>slave_number:</b>	An integer specifying the slave number to use for the custom command.
<b>function_code:</b>	An integer specifying the function code for the custom command.
<b>data:</b>	An array of integers in which each entry must be a valid byte (0-255) value.

**Example command:**

```
modbus_send_custom_command("172.140.17.11", 103, 6,
[17, 32, 2, 88])
```

- Example Parameters:
  - IP address = 172.140.17.11
  - Slave number = 103
  - Function code = 6
  - Data = (17,32,2,88)
    - \* Function code and data are specified by the manufacturer of the slave Modbus device connected to the UR controller



**modbus\_set\_digital\_input\_action**(*signal\_name, action*)

Sets the selected digital input signal to either a "default" or "freedrive" action.

```
>>> modbus_set_digital_input_action("input1", "freedrive")
```

**Parameters**

**signal\_name:** A string identifying a digital input signal that was previously added.

**action:** The type of action. The action can either be "default" or "freedrive". (string)

**Example command:**

```
modbus_set_digital_input_action("input1", "freedrive")
```

- Example Parameters:
  - Signal name = "input1"
  - Action = "freedrive"

**modbus\_set\_output\_register**(*signal\_name, register\_value, is\_secondary\_program*)

Sets the output register signal identified by the given name to the given value.

```
>>> modbus_set_output_register("output1", 300, False)
```

**Parameters**

**signal\_name:** A string identifying an output register signal that in advance has been added.

**register\_value:** An integer which must be a valid word (0-65535) value.

**is\_secondary\_program:** A boolean for internal use only. Must be set to False.

**Example command:** `modbus_set_output_register("output1", 300, False)`

- Example Parameters:
  - Signal name = output1
  - Register value = 300
  - Is\_secondary\_program = False (Note: must be set to False)

**modbus\_set\_output\_signal**(*signal\_name*, *digital\_value*,  
*is\_secondary\_program*)

Sets the output digital signal identified by the given name to the given value.

```
>>> modbus_set_output_signal("output2", True, False)
```

#### Parameters

<code>signal_name:</code>	A string identifying an output digital signal that in advance has been added.
<code>digital_value:</code>	A boolean to which value the signal will be set.
<code>is_secondary_program:</code>	A boolean for internal use only. Must be set to False.

**Example command:** `modbus_set_output_signal("output1", True, False)`

- Example Parameters:
  - Signal name = output1
  - Digital value = True
  - Is\_secondary\_program = False (Note: must be set to False)

**modbus\_set\_signal\_update\_frequency**(*signal\_name*,  
*update\_frequency*)

Sets the frequency with which the robot will send requests to the Modbus controller to either read or write the signal value.

```
>>> modbus_set_signal_update_frequency("output2", 20)
```

#### Parameters

<code>signal_name:</code>	A string identifying an output digital signal that in advance has been added.
<code>update_frequency:</code>	An integer in the range 0-500 specifying the update frequency in Hz.

#### Example command:

`modbus_set_signal_update_frequency("output2", 20)`

- Example Parameters:
  - Signal name = output2
  - Signal update frequency = 20 Hz

**read\_input\_boolean\_register(address)**

Reads the boolean from one of the input registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:127)

**Return Value**

The boolean value held by the register (True, False)

**Note:** The lower range of the boolean input registers (0:63) is reserved for FieldBus/PLC interface usage. The upper range (64:127) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> bool_val = read_input_boolean_register(3)
```

**Example command:** `read_input_boolean_register(3)`

- Example Parameters:
  - Address = input boolean register 3

**read\_input\_float\_register(address)**

Reads the float from one of the input registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:47)

**Return Value**

The value held by the register (float)

**Note:** The lower range of the float input registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> float_val = read_input_float_register(3)
```

**Example command:** `read_input_float_register(3)`

- Example Parameters:
  - Address = input float register 3

**read\_input\_integer\_register(address)**

Reads the integer from one of the input registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:47)

**Return Value**

The value held by the register (-2,147,483,648 : 2,147,483,647)

**Note:** The lower range of the integer input registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> int_val = read_input_integer_register(3)
```

**Example command:** `read_input_integer_register(3)`

- Example Parameters:
  - Address = input integer register 3

**read\_output\_boolean\_register(address)**

Reads the boolean from one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:127)

**Return Value**

The boolean value held by the register (True, False)

**Note:** The lower range of the boolean output registers (0:63) is reserved for FieldBus/PLC interface usage. The upper range (64:127) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> bool_val = read_output_boolean_register(3)
```

**Example command:** `read_output_boolean_register(3)`

- Example Parameters:
  - Address = output boolean register 3

**read\_output\_float\_register(address)**

Reads the float from one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:47)

**Return Value**

The value held by the register (float)

**Note:** The lower range of the float output registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> float_val = read_output_float_register(3)
```

**Example command:** `read_output_float_register(3)`

- Example Parameters:
  - Address = output float register 3

**read\_output\_integer\_register(address)**

Reads the integer from one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

`address`: Address of the register (0:47)

**Return Value**

The int value held by the register (-2,147,483,648 : 2,147,483,647)

**Note:** The lower range of the integer output registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> int_val = read_output_integer_register(3)
```

**Example command:** `read_output_integer_register(3)`

- Example Parameters:
  - Address = output integer register 3

**read\_port\_bit(address)**

Reads one of the ports, which can also be accessed by Modbus clients

```
>>> boolval = read_port_bit(3)
```

**Parameters**

**address:** Address of the port (See port map on Support site, page "Modbus Server" )

**Return Value**

The value held by the port (True, False)

**Example command:** `read_port_bit(3)`

- Example Parameters:
  - Address = port bit 3

**read\_port\_register(address)**

Reads one of the ports, which can also be accessed by Modbus clients

```
>>> intval = read_port_register(3)
```

**Parameters**

**address:** Address of the port (See port map on Support site, page "Modbus Server" )

**Return Value**

The signed integer value held by the port (-32768 : 32767)

**Example command:** `read_port_register(3)`

- Example Parameters:
  - Address = port register 3

**rpc\_factory**(type, url)

Creates a new Remote Procedure Call (RPC) handle. Please read the subsection ef{Remote Procedure Call (RPC)} for a more detailed description of RPCs.

```
>>> proxy = rpc_factory("xmlrpc", "http://127.0.0.1:8080/RPC2")
```

**Parameters**

- type:** The type of RPC backed to use. Currently only the "xmlrpc" protocol is available.
- url:** The URL to the RPC server. Currently two protocols are supported: pstream and http. The pstream URL looks like "<ip-address>:<port>", for instance "127.0.0.1:8080" to make a local connection on port 8080. A http URL generally looks like "http://<ip-address>:<port>/<path>", whereby the <path> depends on the setup of the http server. In the example given above a connection to a local Python webserver on port 8080 is made, which expects XMLRPC calls to come in on the path "RPC2".

**Return Value**

A RPC handle with a connection to the specified server using the designated RPC backend. If the server is not available the function and program will fail. Any function that is made available on the server can be called using this instance. For example "bool isTargetAvailable(int number, ...)" would be "proxy.isTargetAvailable(var\_1, ...)", whereby any number of arguments are supported (denoted by the ...).

**Note:** Giving the RPC instance a good name makes programs much more readable (i.e. "proxy" is not a very good name).

**Example command:** `rpc_factory("xmlrpc", "http://127.0.0.1:8080/RPC2")`

- Example Parameters:
  - type = xmlrpc
  - url = http://127.0.0.1:8080/RPC2

**`rtde_set_watchdog(variable_name, min_frequency, action='pause')`**

This function will activate a watchdog for a particular input variable to the RTDE. When the watchdog did not receive an input update for the specified variable in the time period specified by `min_frequency` (Hz), the corresponding action will be taken. All watchdogs are removed on program stop.

```
>>> rtde_set_watchdog("input_int_register_0", 10, "stop")
```

#### Parameters

- `variable_name`: Input variable name (string), as specified by the RTDE interface
- `min_frequency`: The minimum frequency (float) an input update is expected to arrive.
- `action`: Optional: Either "ignore", "pause" or "stop" the program on a violation of the minimum frequency. The default action is "pause".

#### Return Value

None

**Note:** Only one watchdog is necessary per RTDE input package to guarantee the specified action on missing updates.

**Example command:** `rtde set watchdog( "input int register 0" , 10, "stop" )`

- Example Parameters:
  - variable name = input int register 0
  - min frequency = 10 hz
  - action = stop the program



**set\_analog\_inputrange(port, range)**

*Deprecated:* Set range of analog inputs

Port 0 and 1 is in the controller box, 2 and 3 is in the tool connector.

**Parameters**

- `port`: analog input port number, 0,1 = controller, 2,3 = tool
- `range`: *Controller* analog input range 0: 0-5V (maps automatically onto range 2) and range 2: 0-10V.
- `range`: *Tool* analog input range 0: 0-5V (maps automatically onto range 1), 1: 0-10V and 2: 4-20mA.

**Deprecated:** The `set_standard_analog_input_domain` and `set_tool_analog_input_domain` replace this function. Ports 2-3 should be changed to 0-1 for the latter function. This function might be removed in the next major release.

**Note:** For *Controller* inputs ranges 1: -5-5V and 3: -10-10V are no longer supported and will show an exception in the GUI.

**set\_analog\_out(n, f)**

*Deprecated:* Set analog output signal level

**Parameters**

- `n`: The number (id) of the output, integer: (0:1)
- `f`: The relative signal level (0:1) (float)

**Deprecated:** The `set_standard_analog_out` replaces this function. This function might be removed in the next major release.

**Example command:** `set_analog_out(1, 0.5)`

- Example Parameters:
  - `n` is analog output port 1 (on controller)
  - `f` = 0.5, that corresponds to 5V (or 12mA depending on domain setting) on the output port

**set\_configurable\_digital\_out(*n, b*)**

Set configurable digital output signal level

See also `set_standard_digital_out` and `set_tool_digital_out`.

**Parameters**

- n*: The number (id) of the output, integer: (0:7)
- b*: The signal level. (boolean)

**Example command:** `set_configurable_digital_out(1, True)`

- Example Parameters:
  - *n* is configurable digital output 1
  - *b* = True

**set\_digital\_out(*n, b*)**

*Deprecated:* Set digital output signal level

**Parameters**

- n*: The number (id) of the output, integer: (0:9)
- b*: The signal level. (boolean)

**Deprecated:** The `set_standard_digital_out` and `set_tool_digital_out` replace this function. Ports 8-9 should be changed to 0-1 for the latter function. This function might be removed in the next major release.

**Example command:** `set_digital_out(1, True)`

- Example Parameters:
  - *n* is digital output 1
  - *b* = True

**set\_flag(*n, b*)**

Flags behave like internal digital outputs. They keep information between program runs.

**Parameters**

- n*: The number (id) of the flag, integer: (0:31)
- b*: The stored bit. (boolean)

**Example command:** `set_flag(1, True)`

- Example Parameters:
  - *n* is flag number 1
  - *b* = True will set the bit to True

**set\_standard\_analog\_out(*n*, *f*)**

Set standard analog output signal level

**Parameters**

- n*: The number (id) of the output, integer: (0:1)
- f*: The relative signal level (0;1) (float)

**Example command:** `set_standard_analog_out(1, 1.0)`

- Example Parameters:
  - *n* is standard analog output port 1
  - *f* = 1.0, that corresponds to 10V (or 20mA depending on domain setting) on the output port

**set\_standard\_digital\_out(*n*, *b*)**

Set standard digital output signal level

See also `set_configurable_digital_out` and `set_tool_digital_out`.

**Parameters**

- n*: The number (id) of the output, integer: (0:7)
- b*: The signal level. (boolean)

**Example command:** `set_standard_digital_out(1, True)`

- Example Parameters:
  - *n* is standard digital output 1
  - *f* = True

**set\_tool\_communication**(*enabled, baud\_rate, parity, stop\_bits, rx\_idle\_chars, tx\_idle\_chars*)

This function will activate or deactivate the 'Tool Communication Interface' (TCI). The TCI will enable communication with a external tool via the robots analog inputs hereby avoiding external wiring.

```
>>> set_tool_communication(True, 115200, 1, 2, 1.0, 3.5)
```

#### Parameters

<code>enabled:</code>	Boolean to enable or disable the TCI (string). Valid values: True (enable), False (disable)
<code>baud_rate:</code>	The used baud rate (int). Valid values: 9600, 19200, 38400, 57600, 115200, 1000000, 2000000, 5000000.
<code>parity:</code>	The used parity (int). Valid values: 0 (none), 1 (odd), 2 (even).
<code>stop_bits:</code>	The number of stop bits (int). Valid values: 1, 2.
<code>rx_idle_chars:</code>	Amount of chars the RX unit in the tool should wait before marking a message as over / sending it to the PC (float). Valid values: min=1.0 max=40.0.
<code>tx_idle_chars:</code>	Amount of chars the TX unit in the tool should wait before starting a new transmission since last activity on bus (float). Valid values: min=0.0 max=40.0.

#### Return Value

None

**Note:** Enabling this feature will disable the robot tool analog inputs.

**Example command:** `set_tool_communication(True, 115200, 1, 2, 1.0, 3.5)`

- Example Parameters:
  - enabled = True
  - baud rate = 115200
  - parity = ODD
  - stop bits = 2
  - rx idle time = 1.0
  - tx idle time = 3.5

**set\_tool\_digital\_out(*n, b*)**

Set tool digital output signal level

See also `set_configurable_digital_out` and `set_standard_digital_out`.

For description of operating modes when activated, see Tool Digital Outputs section in the manual.

**Parameters**

- `n`: The number (id) of the output, integer: (0:1)
- `b`: The signal activation. (boolean)

**Example command:** `set_tool_digital_out(1, True)`

- Example Parameters:
  - `n` is tool digital output 1
  - `b` = True

**set\_tool\_digital\_output\_mode(*n, mode*)**

Set output mode of tool output pin

**Parameters**

- `n`: The number (id) of the output, integer: (0:1)
- `mode`: The pin mode. Integer: (1:3). 1 = Sinking/NPN, 2 = Sourcing / PNP, 3 = Push-Pull.

**Example command:** `set_tool_digital_output_mode(0, 2)`

- Example Parameters:
  - 0 is digital output pin 0.
  - 2 is pin mode sourcing/PNP. The pin will source current when set to 1 and be high impedance when set to 0.

**set\_tool\_output\_mode(*mode*)**

Set tool Digital Outputs mode

**Parameters**

- `mode`: The output mode to set, integer: (0:1). 0 = Digital Output mode, 1 = Power(dual pin) mode

**Example command:** `set_tool_output_mode(1)`

- Example Parameters:
  - 1 is power(dual pin) mode. The digital outputs will be used as extra supply.

**set\_tool\_voltage(voltage)**

Sets the voltage level for the power supply that delivers power to the connector plug in the tool flange of the robot. The voltage can be 0, 12 or 24 volts.

**Parameters**

**voltage:** The voltage (as an integer) at the tool connector, integer: 0, 12 or 24.

**Example command:** `set_tool_voltage(24)`

- Example Parameters:
  - voltage = 24 volts

**socket\_close(socket\_name='socket\_0')**

Closes TCP/IP socket communication

Closes down the socket connection to the server.

```
>>> socket_comm_close()
```

**Parameters**

**socket\_name:** Name of socket (string)

**Example command:** `socket_close(socket_name="socket_0")`

- Example Parameters:
  - socket\_name = socket\_0

**socket\_get\_var(name, socket\_name='socket\_0')**

Reads an integer from the server

Sends the message "GET <name>\n" through the socket, expects the response "<name> <int>\n" within 2 seconds. Returns 0 after timeout

**Parameters**

**name:** Variable name (string)

**socket\_name:** Name of socket (string)

**Return Value**

an integer from the server (int), 0 is the timeout value

**Example command:** `x_pos = socket_get_var("POS_X")`

Sends: GET POS\_X\n to socket\_0, and expects response within 2s

- Example Parameters:
  - name = POS\_X → name of variable
  - socket\_name = default: socket\_0

---

**socket\_open**(address, port, socket\_name='socket\_0')

Open TCP/IP ethernet communication socket

Attempts to open a socket connection, times out after 2 seconds.

**Parameters**

address: Server address (string)  
port: Port number (int)  
socket\_name: Name of socket (string)

**Return Value**

False if failed, True if connection successfully established

**Note:** The used network setup influences the performance of client/server communication. For instance, TCP/IP communication is buffered by the underlying network interfaces.

**Example command:** `socket_open("192.168.5.1", 50000, "socket_10")`

- Example Parameters:
  - address = 192.168.5.1
  - port = 50000
  - socket\_name = socket\_10

**socket\_read\_ascii\_float**(*number*, *socket\_name*=' socket\_0' ,  
*timeout*=2)

Reads a number of ascii formatted floats from the socket. A maximum of 30 values can be read in one command.

The format of the numbers should be in parantheses, and seperated by ", ". An example list of four numbers could look like "( 1.414 , 3.14159, 1.616, 0.0)".

The returned list contains the total numbers read, and then each number in succession. For example a read\_ascii\_float on the example above would return (4, 1.414, 3.14159, 1.616, 0.0).

A failed read or timeout will return the list with 0 as first element and then "Not a number (nan)" in the following elements (ex. (0, nan, nan, nan) for a read of three numbers).

#### Parameters

- |                            |   |
|----------------------------|---|
| <code>number :</code>      | The number of variables to read (int)   |
| <code>socket_name :</code> | Name of socket (string)   |
| <code>timeout :</code>     | The number of seconds until the read action times out (float). A timeout of 0 or negative number indicates that the function should not return until a read is completed. |

#### Return Value

A list of numbers read (list of floats, length=number+1)

**Example command:** `list_of_four_floats = socket_read_ascii_float(4, "socket_10")`

- Example Parameters:
  - number = 4 → Number of floats to read
  - socket\_name = socket\_10
  - \* returns list



**socket\_read\_binary\_integer**(number, socket\_name=' socket\_0' ,  
timeout=2)

Reads a number of 32 bit integers from the socket. Bytes are in network byte order. A maximum of 30 values can be read in one command.

Returns (for example) (3,100,2000,30000), if there is a timeout or the reply is invalid, (0,-1,-1,-1) is returned, indicating that 0 integers have been read

#### Parameters

- |              |   |
|--------------|---|
| number:      | The number of variables to read (int)   |
| socket_name: | Name of socket (string)   |
| timeout:     | The number of seconds until the read action times out (float). A timeout of 0 or negative number indicates that the function should not return until a read is completed. |

#### Return Value

A list of numbers read (list of ints, length=number+1)

**Example command:** list\_of\_ints =  
socket\_read\_binary\_integer(4, "socket\_10")

- Example Parameters:
  - number = 4 → Number of integers to read
  - socket\_name = socket\_10

**`socket_read_byte_list`**(*number*, *socket\_name*='socket\_0', *timeout*=2)

Reads a number of bytes from the socket. A maximum of 30 values can be read in one command.

Returns (for example) (3,100,200,44), if there is a timeout or the reply is invalid, (0,-1,-1,-1) is returned, indicating that 0 bytes have been read

#### Parameters

- `number`: The number of bytes to read (int)
- `socket_name`: Name of socket (string)
- `timeout`: The number of seconds until the read action times out (float). A timeout of 0 or negative number indicates that the function should not return until a read is completed.

#### Return Value

A list of numbers read (list of ints, length=number+1)

**Example command:** `list_of_bytes = socket_read_byte_list(4, "socket_10")`

- Example Parameters:
  - `number = 4` → Number of byte variables to read
  - `socket_name = socket_10`

**socket\_read\_line**(*socket\_name*=' socket\_0' , *timeout*=2)

*Deprecated:* Reads the socket buffer until the first "\r\n" (carriage return and newline) characters or just the "\n" (newline) character, and returns the data as a string. The returned string will not contain the "\n" nor the "\r\n" characters.

Returns (for example) "reply from the server:", if there is a timeout or the reply is invalid, an empty line is returned (""). You can test if the line is empty with an if-statement.

```
>>> if(line_from_server) :  
>>>     popup("the line is not empty")  
>>> end
```

**Parameters**

*socket\_name*: Name of socket (string)

*timeout*: The number of seconds until the read action times out (float). A timeout of 0 or negative number indicates that the function should not return until a read is completed.

**Return Value**

One line string

**Deprecated:** The `socket_read_string` replaces this function. Set flag "interpret\_escape" to "True" to enable the use of escape sequences "\n" "\r" and "\t" as a prefix or suffix.

**Example command:** `line_from_server = socket_read_line("socket_10")`

- Example Parameters:
  - `socket_name = socket_10`

---

```
socket_read_string(socket_name=' socket_0' , prefix=' ' , suffix=' ' ,
interpret_escape=' False' , timeout=2)
```

---

Reads all data from the socket and returns the data as a string.

Returns (for example) "reply from the server:\n Hello World". if there is a timeout or the reply is invalid, an empty string is returned (""). You can test if the string is empty with an if-statement.

```
>>> if(string_from_server) :
>>>     popup ("the string is not empty")
>>> end
```

The optional parameters "prefix" and "suffix", can be used to express what is extracted from the socket. The "prefix" specifies the start of the substring (message) extracted from the socket. The data up to the end of the "prefix" will be ignored and removed from the socket. The "suffix" specifies the end of the substring (message) extracted from the socket. Any remaining data on the socket, after the "suffix", will be preserved.

By using the "prefix" and "suffix" it is also possible send multiple string to the controller at once, because the suffix defines where the message ends. E.g. sending ">hello<>world<" and calling this script function with the prefix=">" and suffix="<".

Note that leading spaces in the prefix and suffix strings are ignored in the current software and may cause communication errors in future releases.

The optional parameter "interpret\_escape" can be used to allow the use of escape sequences "\n", "\t" and "\r" as part of the prefix or suffix.

#### Parameters

socket_name:	Name of socket (string)
prefix:	Defines a prefix (string)
suffix:	Defines a suffix (string)
interpret_escape:	Enables the interpretation of escape sequences (bool)
timeout:	The number of seconds until the read action times out (float). A timeout of 0 or negative number indicates that the function should not return until a read is completed.

#### Return Value

String

**Example command:** string\_from\_server =  
socket\_read\_string("socket\_10",prefix=">",suffix="<")

---

**socket\_send\_byte**(*value*, *socket\_name*=' socket\_0')

Sends a byte to the server

Sends the byte <value> through the socket. Expects no response. Can be used to send special ASCII characters: 10 is newline, 2 is start of text, 3 is end of text.

**Parameters**

*value*:                   The number to send (byte)  
*socket\_name*:   Name of socket (string)

**Return Value**

a boolean value indicating whether the send operation was successful

**Example command:** `socket_send_byte(2, "socket_10")`

- Example Parameters:
  - `value = 2`
  - `socket_name = socket_10`
  - \* Returns True or False (sent or not sent)

---

**socket\_send\_int**(*value*, *socket\_name*=' socket\_0')

Sends an int (int32\_t) to the server

Sends the int <value> through the socket. Send in network byte order. Expects no response.

**Parameters**

*value*:                   The number to send (int)  
*socket\_name*:   Name of socket (string)

**Return Value**

a boolean value indicating whether the send operation was successful

**Example command:** `socket_send_int(2, "socket_10")`

- Example Parameters:
  - `value = 2`
  - `socket_name = socket_10`
  - \* Returns True or False (sent or not sent)

**socket\_send\_line**(*str*, *socket\_name*=' socket\_0')

Sends a string with a newline character to the server - useful for communicating with the UR dashboard server

Sends the string <str> through the socket in ASCII coding. Expects no response.

**Parameters**

*str*: The string to send (ascii)  
*socket\_name*: Name of socket (string)

**Return Value**

a boolean value indicating whether the send operation was successful

**Example command:** `socket_send_line("hello", "socket_10")`

Sends: hello\n to socket\_10

- Example Parameters:
  - *str* = hello
  - *socket\_name* = socket\_10
- \* Returns True or False (sent or not sent)

**socket\_send\_string**(*str*, *socket\_name*=' socket\_0')

Sends a string to the server

Sends the string <str> through the socket in ASCII coding. Expects no response.

**Parameters**

*str*: The string to send (ascii)  
*socket\_name*: Name of socket (string)

**Return Value**

a boolean value indicating whether the send operation was successful

**Example command:** `socket_send_string("hello", "socket_10")`

Sends: hello to socket\_10

- Example Parameters:
  - *str* = hello
  - *socket\_name* = socket\_10
- \* Returns True or False (sent or not sent)

---

**socket\_set\_var**(name, value, socket\_name='socket\_0')

Sends an integer to the server

Sends the message "SET <name> <value>\n" through the socket.  
Expects no response.

**Parameters**

name: Variable name (string)  
value: The number to send (int)  
socket\_name: Name of socket (string)

**Example command:** `socket_set_var("POS_Y", 2200, "socket_10")`

Sends string: SET POS\_Y 2200\n to socket\_10

- Example Parameters:
  - name = POS\_Y → name of variable
  - value = 2200
  - socket\_name = socket\_10

---

**write\_output\_boolean\_register**(address, value)

Writes the boolean value into one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

address: Address of the register (0:127)  
value: Value to set in the register (True, False)

**Note:** The lower range of the boolean output registers (0:63) is reserved for FieldBus/PLC interface usage. The upper range (64:127) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> write_output_boolean_register(3, True)
```

**Example command:** `write_output_boolean_register(3, True)`

- Example Parameters:
  - address = 3
  - value = True

**write\_output\_float\_register**(address, value)

Writes the float value into one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

address: Address of the register (0:47)

value: Value to set in the register (float)

**Note:** The lower part of the float output registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> write_output_float_register(3, 37.68)
```

**Example command:** write\_output\_float\_register(3,37.68)

- Example Parameters:
  - address = 3
  - value = 37.68

**write\_output\_integer\_register**(address, value)

Writes the integer value into one of the output registers, which can also be accessed by a Field bus. Note, uses it's own memory space.

**Parameters**

address: Address of the register (0:47)

value: Value to set in the register (-2,147,483,648 : 2,147,483,647)

**Note:** The lower range of the integer output registers (0:23) is reserved for FieldBus/PLC interface usage. The upper range (24:47) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

```
>>> write_output_integer_register(3, 12)
```

**Example command:** write\_output\_integer\_register(3,12)

- Example Parameters:
  - address = 3
  - value = 12



**write\_port\_bit(address, value)**

Writes one of the ports, which can also be accessed by Modbus clients

```
>>> write_port_bit(3, True)
```

**Parameters**

**address:** Address of the port (See port map on Support site, page "Modbus Server" )

**value:** Value to be set in the register (True, False)

**Example command:** `write_port_bit(3, True)`

- Example Parameters:
  - Address = 3
  - Value = True

**write\_port\_register(address, value)**

Writes one of the ports, which can also be accessed by Modbus clients

```
>>> write_port_register(3, 100)
```

**Parameters**

**address:** Address of the port (See port map on Support site, page "Modbus Server" )

**value:** Value to be set in the port (0 : 65536) or (-32768 : 32767)

**Example command:** `write_port_register(3, 100)`

- Example Parameters:
  - Address = 3
  - Value = 100

**zero\_ftsensor()**

Zeroes the TCP force/torque measurement from the builtin force/torque sensor by subtracting the current measurement from the subsequent.

**Note:** Function only applicable in G5

## 5.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> None

## 6 Module ioconfiguration

### 6.1 Functions

**modbus\_set\_runstate\_dependent\_choice**(*signal\_name*,  
*runstate\_choice*)

Sets the output signal levels depending on the state of the program (running or stopped).

```
>>> modbus_set_runstate_dependent_choice("output2", 1)
```

**Parameters**

**signal\_name:** A string identifying an output digital signal that in advance has been added.

**runstate\_choice:** An integer: 0 = preserve program state, 1 = set low when a program is not running, 2 = set high when a program is not running, 3 = High when program is running and low when it is stopped.

**Example command:**

```
modbus_set_runstate_dependent_choice("output2", 3)
```

- Example Parameters:
  - Signal name = output2
  - Runstate dependent choice = 3 → set low when a program is stopped and high when a program is running

**set\_analog\_outputdomain**(*port*, *domain*)

Set domain of analog outputs

**Parameters**

**port:** analog output port number

**domain:** analog output domain: 0: 4-20mA, 1: 0-10V

**Example command:** `set_analog_outputdomain(1,1)`

- Example Parameters:
  - port is analog output port 1 (on controller)
  - domain = 1 (0-10 volts)

**set\_configurable\_digital\_input\_action**(*port, action*)

Using this method sets the selected configurable digital input register to either a "default" or "freedrive" action.

See also:

- `set_input_actions_to_default`
- `set_standard_digital_input_action`
- `set_tool_digital_input_action`
- `set_gp_boolean_input_action`

**Parameters**

`port`: The configurable digital input port number.  
(integer)

`action`: The type of action. The action can either be  
"default" or "freedrive". (string)

**Example command:** `set_configurable_digital_input_action(0, "freedrive")`

- Example Parameters:
  - `n` is the configurable digital input register 0
  - `f` is set to "freedrive" action

**set\_gp\_boolean\_input\_action(port, action)**

Using this method sets the selected gp boolean input register to either a "default" or "freedrive" action.

**Parameters**

- port:** The gp boolean input port number. integer: (0:127)
- action:** The type of action. The action can either be "default" or "freedrive". (string)

**Note:** The lower range of the boolean input registers (0:63) is reserved for FieldBus/PLC interface usage. The upper range (64:127) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

See also:

- set\_input\_actions\_to\_default
- set\_standard\_digital\_input\_action
- set\_configurable\_digital\_input\_action
- set\_tool\_digital\_input\_action

**Example command:** set\_gp\_boolean\_input\_action(64, "freedrive")

- Example Parameters:
  - n is the gp boolean input register 0
  - f is set to "freedrive" action

**set\_input\_actions\_to\_default()**

Using this method sets the input actions of all standard, configurable, tool, and gp\_boolean input registers to "default" action.

See also:

- set\_standard\_digital\_input\_action
- set\_configurable\_digital\_input\_action
- set\_tool\_digital\_input\_action
- set\_gp\_boolean\_input\_action

**Example command:** set\_input\_actions\_to\_default()

**set\_runstate\_configurable\_digital\_output\_to\_value(outputId, state)**

Sets the output signal levels depending on the state of the program (running or stopped).

Example: Set configurable digital output 5 to high when program is not running.

```
>>> set_runstate_configurable_digital_output_to_value(5, 2)
```

**Parameters**

**outputId:** The output signal number (id), integer: (0:7)

**state:** The state of the output, integer: 0 = Preserve state, 1 = Low when program is not running, 2 = High when program is not running, 3 = High when program is running and low when it is stopped.

**Example command:**

```
set_runstate_configurable_digital_output_to_value(5, 2)
```

- Example Parameters:
  - outputid = configurable digital output on port 5
  - Runstate choice = 2 → High when program is not running

**set\_runstate\_gp\_boolean\_output\_to\_value(outputId, state)**

Sets the output value depending on the state of the program (running or stopped).

**Parameters**

**outputId:** The output signal number (id), integer: (0:127)

**state:** The state of the output, integer: 0 = Preserve state, 1 = Low when program is not running, 2 = High when program is not running, 3 = High when program is running and low when it is stopped.

**Note:** The lower range of the boolean output registers (0:63) is reserved for FieldBus/PLC interface usage. The upper range (64:127) cannot be accessed by FieldBus/PLC interfaces, since it is reserved for external RTDE clients.

**Example command:**

```
set_runstate_gp_boolean_output_to_value(64, 2)
```

- Example Parameters:
  - outputid = output on port 64
  - Runstate choice = 2 → High when program is not running

**set\_runstate\_standard\_analog\_output\_to\_value(outputId, state)**

Sets the output signal levels depending on the state of the program (running or stopped).

Example: Set standard analog output 1 to high when program is not running.

```
>>> set_runstate_standard_analog_output_to_value(1, 2)
```

**Parameters**

**outputId:** The output signal number (id), integer: (0:1)  
**state:** The state of the output, integer: 0 = Preserve state, 1 = Min when program is not running, 2 = Max when program is not running, 3 = Max when program is running and Min when it is stopped.

**Example command:**

```
set_runstate_standard_analog_output_to_value(1, 2)
```

- Example Parameters:
  - outputid = standard analog output on port 1
  - Runstate choice = 2 → High when program is not running

**set\_runstate\_standard\_digital\_output\_to\_value(outputId, state)**

Sets the output signal level depending on the state of the program (running or stopped).

Example: Set standard digital output 5 to high when program is not running.

```
>>> set_runstate_standard_digital_output_to_value(5, 2)
```

**Parameters**

**outputId:** The output signal number (id), integer: (0:7)  
**state:** The state of the output, integer: 0 = Preserve state, 1 = Low when program is not running, 2 = High when program is not running, 3 = High when program is running and low when it is stopped.

**Example command:**

```
set_runstate_standard_digital_output_to_value(5, 2)
```

- Example Parameters:
  - outputid = standard digital output on port 1
  - Runstate choice = 2 → High when program is not running

**set\_runstate\_tool\_digital\_output\_to\_value(outputId, state)**

Sets the output signal level depending on the state of the program (running or stopped).

Example: Set tool digital output 1 to high when program is not running.

```
>>> set_runstate_tool_digital_output_to_value(1, 2)
```

**Parameters**

outputId: The output signal number (id), integer: (0:1)

state: The state of the output, integer: 0 = Preserve state, 1 = Low when program is not running, 2 = High when program is not running, 3 = High when program is running and low when it is stopped.

**Example command:**

```
set_runstate_tool_digital_output_to_value(1, 2)
```

- Example Parameters:
  - outputid = tool digital output on port 1
  - Runstate choice = 2 → High when program is not running

**set\_standard\_analog\_input\_domain(port, domain)**

Set domain of standard analog inputs in the controller box

For the tool inputs see `set_tool_analog_input_domain`.

**Parameters**

port: analog input port number: 0 or 1

domain: analog input domains: 0: 4-20mA, 1: 0-10V

**Example command:** `set_standard_analog_input_domain(1,0)`

- Example Parameters:
  - port = analog input port 1
  - domain = 0 (4-20 mA)

**set\_standard\_digital\_input\_action(*port, action*)**

Using this method sets the selected standard digital input register to either a "default" or "freedrive" action.

See also:

- `set_input_actions_to_default`
- `set_configurable_digital_input_action`
- `set_tool_digital_input_action`
- `set_gp_boolean_input_action`

**Parameters**

- `port`: The standard digital input port number. (integer)
- `action`: The type of action. The action can either be "default" or "freedrive". (string)

**Example command:** `set_standard_digital_input_action(0, "freedrive")`

- Example Parameters:
  - `n` is the standard digital input register 0
  - `f` is set to "freedrive" action

**set\_tool\_analog\_input\_domain(*port, domain*)**

Set domain of analog inputs in the tool

For the controller box inputs see `set_standard_analog_input_domain`.

**Parameters**

- `port`: analog input port number: 0 or 1
- `domain`: analog input domains: 0: 4-20mA, 1: 0-10V

**Example command:** `set_tool_analog_input_domain(1, 1)`

- Example Parameters:
  - `port` = tool analog input 1
  - `domain` = 1 (0-10 volts)



**set\_tool\_digital\_input\_action(port, action)**

Using this method sets the selected tool digital input register to either a "default" or "freedrive" action.

See also:

- set\_input\_actions\_to\_default
- set\_standard\_digital\_input\_action
- set\_configurable\_digital\_input\_action
- set\_gp\_boolean\_input\_action

**Parameters**

- port:** The tool digital input port number. (integer)
- action:** The type of action. The action can either be "default" or "freedrive". (string)

**Example command:** `set_tool_digital_input_action(0, "freedrive")`

- Example Parameters:
  - n is the tool digital input register 0
  - f is set to "freedrive" action

## 6.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> None

## 7 Module processpath

This module contains Remote TCP and toolpath APIs.

**NOTE:**

1. These script functions are only accessible if the program is being run from within Polyscope.
2. The Remote TCP & Toolpath URCap Controller must run in order for the script functions to work. Go to Installation > URCaps > Remote TCP & Toolpath URCap to start the controller from the URCap home page.
3. The script functions are available in the Functions dropdown within the Script node (Program > Advanced > Script). Note: `mc_add_circular()` requires six arguments, but only five arguments are displayed in the Functions dropdown. Please make sure to specify the sixth argument 'mode' when using this function.

## 7.1 Functions

**`mc_add_circular(pose_via, pose_to, a, v, r, mode=0)`**

Add a motion to move to pose (circular in tool-space).

This command queues a motion that interpolates circular movement of the regular TCP (if called after `mc_initialize` with `mode = 0`) or the remote TCP (if called after `mc_initialize` with `mode = 1`) to a pose in the Part Coordinate System (PCS).

### Parameters

- `pose_via`: path point (note: only position is used) in the PCS
- `pose_to`: target pose (note: only position is used in Fixed orientation mode) in the PCS
- `a`: tool acceleration ( $\text{m/s}^2$ )
- `v`: tool speed ( $\text{m/s}$ )
- `r`: blend radius (of target pose) ( $\text{m}$ )
- `mode`:  
 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)

### Return Value

The motion ID

**Example command:** `id = mc_add_circular(p[x, y, z, 0, 0, 0], pose_to, a=1.2, v=0.25, r=0.05, mode=1)`

- Example Parameters:
  - Note: first position on circle is previous waypoint.
  - `pose_via = p(x, y, z, 0, 0, 0)` → second position on circle.  
 \* Note: rotations are not used so they can be left as zeros.
  - `pose_to` → third (and final) position on circle
  - `a = 1.2` → acceleration is  $1.2 \text{ m/s}^2$
  - `v = 0.25` → velocity is  $250 \text{ mm/s}$
  - `r = 0` → blend radius (at `pose_to`) is  $50 \text{ mm}$
  - `mode = 1` → use fixed orientation relative to tangent of circular arc

**mc\_add\_linear(*pose, a, v, r*)**

Add a motion to move to pose (linear in tool-space).

This command queues a motion that interpolates linear movement of the regular TCP (if called after `mc_initialize` with `mode = 0`) or the remote TCP (if called after `mc_initialize` with `mode = 1`) to a pose in the Part Coordinate System (PCS).

**Parameters**

- `pose`: target pose in the PCS
- `a`: tool acceleration (m/s<sup>2</sup>)
- `v`: tool speed (m/s)
- `r`: blend radius (m)

**Return Value**

The motion ID

**Example command:** `id = mc_add_linear(pose, a=1.2, v=0.25, r=0)`

- Example Parameters:
  - `pose = p(0.2, 0.3, 0.5, 0, 0, 3.14)` → position in PCS is `x = 200mm, y = 300mm, z = 500mm, rx = 0, ry = 0, rz = 180deg`.
  - `a = 1.2` → acceleration is 1.2 m/s<sup>2</sup>
  - `v = 0.25` → velocity is 250 mm/s
  - `r = 0` → the blend radius is 0 meters

**mc\_add\_path(path\_id, a, v, r)**

Add a motion to move according to the g-codes specified in the nc\_file.

**Parameters**

path\_id: path ID returned from the mc\_load\_path call  
a: tool acceleration (m/s<sup>2</sup>)  
v: tool speed (m/s)  
r: blend radius (m)

**Return Value**

The motion ID

**Example command:** id = mc\_add\_path(path\_id=1, a=1.2, v=0.25, r=0)

- Example Parameters:
  - path\_id = 1 → path\_id from the mc\_load\_path call
  - a = 1.2 → acceleration is 1.2 m/s<sup>2</sup>
  - v = 0.25 → velocity is 250 mm/s. This value is ignored if useFeedRate=True when calling mc\_load\_path
  - r = 0 → the blend radius is 0 meters

**mc\_get\_target\_rtcp\_speed()**

This command returns the target linear speed of a Remote TCP. To get the target linear speed of a regular TCP, please use get\_target\_tcp\_speed\_along\_path().

**Return Value**

Remote TCP speed in m/s. This is a scalar without any indication of travel direction.

**Example command:** rtcpSpeed = mc\_get\_target\_rtcp\_speed()

**mc\_initialize(mode, tcp, doc=6)**

Initialize a new motion sequence.

This command must be called before every TCP motion sequence. The command resets the speed factor to 1 (see also `mc_set_speed_factor`), the Part Coordinate System to identity (see also `mc_set_pcs`), and the motion ID counter to 0. The command automatically turns on path divergence check if not in simulation mode.

**Parameters**

- `mode`: 0: Initialize regular TCP motion.  
1: Initialize remote TCP motion.
- `tcp`: If `mode = 0`, the regular TCP offset, i.e. the transformation from the output flange coordinate system to the TCP as a pose.  
If `mode = 1`, the remote TCP offset, i.e. the transformation from the robot base to the remote TCP as a pose.
- `doc`: degree of constraints  
6: Constrain all 6-Degree of Freedom (DOF) of the tool.  
5: Constrain 5-DOF of the tool. The rotation about z-axis is not constrained.

**Example command:** `mc_initialize(1, p[0, 0.2, 0.3, 0, 3.14, 0], doc=6)`

- Example Parameters:
  - `mode = 1` → remote TCP motion
  - `tcp = p(0, 0.2, 0.3, 0, 3.14, 0)` → tool center point is set to x = 0mm, y = 200mm, z = 300mm, rotation vector is rx = 0deg, ry = 180deg, rz = 0deg, in robot base coordinates.
  - `doc = 6` → constrain 6-DOF

**mc\_load\_path(nc\_file, useFeedRate)**

Load a .nc toolpath file. This function returns a path id which is to be used with the subsequent mc\_add\_path call. This function has to be called for the same nc\_file if it is to be executed with different speed and acceleration values.

**Parameters**

- nc\_file: the .nc file on the disk
- useFeedRate: specify whether to use feedrate mentioned in the NC file

**Return Value**

The path ID

**Example command:** path\_id =  
mc\_load\_path("/programs/path.nc", useFeedRate=True)

- Example Parameters:
  - nc\_file = /programs/path.nc → the file that contains GCode is /programs/path.nc
  - useFeedRate = True → use the feedrate from .nc file

**mc\_run\_motion(id=-1)**

Run through the motion with ID id.

This command starts to move the robot (if not already started) then waits for the motion with ID = id to finish. A motion with ID = x will only be executed if mc\_run\_motion is called on id >= x or -1. This command is used to run all queued motions (if id = -1) or when users would like to call some instantaneous script (such as set\_standard\_digital\_out) after a particular motion. Do not call commands that take time to finish (such as movep). Doing so will likely cause path divergence error.

**Parameters**

- id: the motion ID. If id = -1, the command will run all motions queued. If id = x, where x >= 0, the command will run all motions with ID <= x.

**Example command:** mc\_run\_motion(id=2)

- Example Parameters:
  - id = 2 → run motion with ID = 0, 1 and 2 to finish.

**mc\_set\_pcs(pcs)**

Set the Part Coordinate System (PCS).

This command sets the PCS either in robot base coordinates (if called after `mc_initialize` with `mode = 0`) or in tool flange coordinates (if called after `mc_initialize` with `mode = 1`). Subsequent poses/paths of motions will be relative to this PCS.

**Parameters**

`pcs`: The part coordinate system as a pose.

**Example command:** `mc_set_pcs(p[0, 0.2, 0.3, 0, 3.14, 0])`

- Example Parameters:
  - `pcs = p(0, 0.2, 0.3, 0, 3.14, 0)` → part coordinate system is set to `x = 0mm`, `y = 200mm`, `z = 300mm`, rotation vector is `rx = 0deg`, `ry = 180deg`, `rz = 0deg`.

**mc\_set\_speed\_factor(s)**

Set speed factor in range (0, 1). The tool speed will be  $v \times s$ , where  $v$  is the commanded tool speed in each motion.

This command only affect motions in this module (such as `mc_add_linear`). The command can be used on a separate thread during motion to dynamically change tool speed.

**Example command:** `mc_set_speed_factor(s=0.1)`

- Example Parameters:
  - `s = 0.1` → speed factor is 0.1

## 7.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'ProcessPath'

## 7.3 Examples

Examples on using the Remote TCP and Toolpath Application.

### 7.3.1 Example 1 - Executing a remote TCP path

```
# Initialize with remote TCP (mode = 1) and constrain 6-DOF
```

```

remote_tcp = p[-0.3909, -0.519, 0.295, 1.57, 0.0, 0.0]
pcs = p[0.0, 0.0, 0.01, 0.0, 0.0, 0.0]
mc_initialize(1, remote_tcp, doc = 6)

# Set the PCS Location
mc_set_pcs(pcs)

waypoint1 = p[0.05, 0.1, 0.1, 0.0, 0.0, 0.0]
waypoint2 = p[0.05, 0.05, 0.1, 0.0, 0.0, 0.0]
circle_viapoint = p[0.075, 0.1, 0.1, 0.0, 0.0, 0.0]
circle_endpoint = p[0.1, 0.05, 0.1, 0.0, 0.0, 0.0]
acc_factor = 0.05
speed_factor = 0.25
blend_radius = 0.02
# Circular motion in constrained mode
mode=1

# Add motions
mc_add_linear(waypoint1, acc_factor, speed_factor, blend_radius)
mc_add_linear(waypoint2, acc_factor, speed_factor, blend_radius)
mc_add_circular(circle_viapoint, circle_endpoint, acc_factor,
               speed_factor, blend_radius, mode)

# Set different PCS for Subsequent motion
pcs = p[0.0, 0.0, 0.02, 0.0, 0.0, 0.0]
acc_factor = 0.1
speed_factor = 1.0

mc_set_pcs(pcs)
mc_add_path("/programs/myToolPathFile.nc", acc_factor, speed_factor, 0.0)

# Execute the motions
mc_run_motion()

```

### 7.3.2 Example 2 - Executing a regular TCP path using 5-DOF

```

# Initialize with regular TCP (mode = 0) and constrain 5-DOF
remote_tcp = p[0.0, 0.03, 0.125, 2.3580, 0.0, 0.0]
pcs = p[-0.25, -0.65, 0.05, 0.0, 0.0, -1.57]
mc_initialize(0, remote_tcp, doc = 5)

# Set PCS and add the toolpath motion
waypoint1 = p[0.05, 0.1, 0.1, 0.0, 0.0, 0.0]
speed_factor = 0.1
acc_factor = 0.25
blend_radius = 0.02
blend_rad_toolpath = 0.0
mc_set_pcs(pcs)

```



```
mc_add_linear(waypoint1, acc_factor, speed_factor, blend_radius)
path_id = mc_load_path("/programs/myToolPathFile.nc", useFeedRate = False)
mc_add_path(path_id, acc_factor, speed_factor, blend_rad_toolpath)

# Execute the motions
mc_run_motion()
```

### 7.3.3 Example 3 - Executing a path and set digital output

```
# This examples demonstrates using motion id to wait for a specific motion to
# finish and set digital output

# Initialize with remote TCP (mode = 1) and constrain 6-DOF
remote_tcp = p[-0.3909, -0.519, 0.295, 1.57, 0.0, 0.0]
pcs = p[0.0, 0.0, 0.01, 0.0, 0.0, 0.0]
mc_initialize(1, remote_tcp, doc = 6)

# Set PCS and add motions
mc_set_pcs(pcs)
waypoint1 = p[0.05, 0.1, 0.1, 0.0, 0.0, 0.0]
waypoint2 = p[0.05, 0.05, 0.1, 0.0, 0.0, 0.0]
circle_viapoint1 = p[0.075, 0.1, 0.1, 0.0, 0.0, 0.0]
circle_endpoint1 = p[0.1, 0.05, 0.1, 0.0, 0.0, 0.0]
speed_factor = 0.25
acc_factor = 0.05
blend_radius = 0.02
id0 = mc_add_linear(waypoint1, acc_factor, speed_factor, blend_radius)
id1 = mc_add_linear(waypoint2, acc_factor, speed_factor, blend_radius)
id2 = mc_add_circular(circle_viapoint1, circle_endpoint1,
    acc_factor, speed_factor, blend_radius, mode = 1)
pcs_toolpath = p[0.0, 0.0, 0.02, 0.0, 0.0, 0.0]
mc_set_pcs(pcs_toolpath)
path_id = mc_load_path("/programs/myToolPathFile.nc", useFeedRate = True)
id3 = mc_add_path(path_id, 1.0, 0.0, 0.0)

# Execute motions to id=id2 and wait for id2 to finish.
# This will execute motions from id0 to id2.
mc_run_motion(id2)

# Set digital output
set_standard_digital_out(7, True)

# Execute the rest of the motions, in this case, just id3
mc_run_motion()

# Set digital output
set_standard_digital_out(7, False)
```

## 7.4 myToolPathFile.nc

```
%PM0
N10 G74 Z-5. L1
N20 G7
N30 G54

N40 ; Operation 1
N50 S1000 T1 M6 ; (Bull Mill, Diameter: 10, Flute Length: 25,
; Shoulder Length: 40, Overall Length: 40, Length including Holder: 40,
; Corner Radius: 2, Corner Radius Type: Corner)
N60 M3
N70 G74 Z-5. L1
N80 G54 I0
N90 G7
N100 M8
N110 G01 X-39.5178 Y-0.0003 Z24.9995 C0.0000 B0.0000 F1500
N120 G01 X-38.2246 Y-1.9469 Z24.9995 C0.0000 B0.0000 F1500
N130 G01 X-36.9313 Y-3.8934 Z24.9995 C0.0000 B0.0000 F1500
N140 G01 X-35.6381 Y-5.8400 Z24.9995 C0.0000 B0.0000 F1500
N150 G01 X-34.3448 Y-7.7866 Z24.9995 C0.0000 B0.0000 F1500
N160 G01 X-33.0515 Y-9.7332 Z24.9995 C0.0000 B0.0000 F1500
N170 G01 X-31.7583 Y-11.6798 Z24.9995 C0.0000 B0.0000 F1500
N180 G01 X-30.4650 Y-13.6264 Z24.9995 C0.0000 B0.0000 F1500
N190 G01 X-29.1718 Y-15.5729 Z24.9995 C0.0000 B0.0000 F1500
N200 G01 X-27.8785 Y-17.5195 Z24.9995 C0.0000 B0.0000 F1500
N210 G01 X-26.5853 Y-19.4661 Z24.9995 C0.0000 B0.0000 F1500
N220 G01 X-25.2920 Y-21.4127 Z24.9995 C0.0000 B0.0000 F1500
N230 G01 X-23.9988 Y-23.3593 Z24.9995 C0.0000 B0.0000 F1500
N240 G01 X-22.7055 Y-25.3059 Z24.9995 C0.0000 B0.0000 F1500
N250 G01 X-21.4123 Y-27.2524 Z24.9995 C0.0000 B0.0000 F1500
N260 G01 X-20.1190 Y-29.1990 Z24.9995 C0.0000 B0.0000 F1500
N270 G01 X-18.8258 Y-31.1456 Z24.9995 C0.0000 B0.0000 F1500
N280 G01 X-17.5325 Y-33.0922 Z24.9995 C0.0000 B0.0000 F1500
N290 G01 X-16.2393 Y-35.0388 Z24.9995 C0.0000 B0.0000 F1500
N300 G01 X-14.9460 Y-36.9854 Z24.9995 C0.0000 B0.0000 F1500
N310 G01 X-13.6527 Y-38.9319 Z24.9995 C0.0000 B0.0000 F1500
N320 G01 X-12.3595 Y-40.8785 Z24.9995 C0.0000 B0.0000 F1500
N330 G01 X-11.0662 Y-42.8251 Z24.9995 C0.0000 B0.0000 F1500
N340 G01 X-9.7730 Y-44.7717 Z24.9995 C0.0000 B0.0000 F1500
N350 G01 X-8.4797 Y-46.7183 Z24.9995 C0.0000 B0.0000 F1500
N360 G01 X-7.1865 Y-48.6649 Z24.9995 C0.0000 B0.0000 F1500
N370 G01 X-5.1870 Y-47.5105 Z24.9995 C0.0000 B0.0000 F1500
N380 G01 X-3.1876 Y-46.3561 Z24.9995 C0.0000 B0.0000 F1500
N390 G01 X-1.1882 Y-45.2017 Z24.9995 C0.0000 B0.0000 F1500
N400 G01 X0.8113 Y-44.0473 Z24.9995 C0.0000 B0.0000 F1500
N410 G01 X2.8107 Y-42.8929 Z24.9995 C0.0000 B0.0000 F1500
N420 G01 X4.8102 Y-41.7385 Z24.9995 C0.0000 B0.0000 F1500
N430 G01 X6.8096 Y-40.5841 Z24.9995 C0.0000 B0.0000 F1500
```

```
N440 G01 X8.8090 Y-39.4297 Z24.9995 C0.0000 B0.0000 F1500
N450 G01 X10.8085 Y-38.2753 Z24.9995 C0.0000 B0.0000 F1500
N460 G01 X12.8079 Y-37.1210 Z24.9995 C0.0000 B0.0000 F1500
N470 G01 X14.8074 Y-35.9666 Z24.9995 C0.0000 B0.0000 F1500
N480 G01 X16.8068 Y-34.8122 Z24.9995 C0.0000 B0.0000 F1500
N490 G01 X18.8062 Y-33.6578 Z24.9995 C0.0000 B0.0000 F1500
N500 G01 X20.8057 Y-32.5034 Z24.9995 C0.0000 B0.0000 F1500
N510 G01 X22.8051 Y-31.3490 Z24.9995 C0.0000 B0.0000 F1500
N520 G01 X24.8046 Y-30.1946 Z24.9995 C0.0000 B0.0000 F1500
N530 G01 X26.8040 Y-29.0402 Z24.9995 C0.0000 B0.0000 F1500
N540 G01 X28.8034 Y-27.8858 Z24.9995 C0.0000 B0.0000 F1500
N550 G01 X30.8029 Y-26.7314 Z24.9995 C0.0000 B0.0000 F1500
N560 G01 X32.8023 Y-25.5770 Z24.9995 C0.0000 B0.0000 F1500
N570 G01 X34.8018 Y-24.4226 Z24.9995 C0.0000 B0.0000 F1500
N580 G01 X34.8018 Y-22.0967 Z24.9995 C0.0000 B0.0000 F1500
N590 G01 X34.8017 Y-19.7707 Z24.9995 C0.0000 B0.0000 F1500
N600 G01 X34.8017 Y-17.4447 Z24.9995 C0.0000 B0.0000 F1500
N610 G01 X34.8017 Y-15.1187 Z24.9995 C0.0000 B0.0000 F1500
N620 G01 X34.8017 Y-12.7927 Z24.9995 C0.0000 B0.0000 F1500
N630 G01 X34.8017 Y-10.4667 Z24.9995 C0.0000 B0.0000 F1500
N640 G01 X34.8017 Y-8.1407 Z24.9995 C0.0000 B0.0000 F1500
N650 G01 X34.8017 Y-5.8148 Z24.9995 C0.0000 B0.0000 F1500
N660 G01 X34.8017 Y-3.4888 Z24.9995 C0.0000 B0.0000 F1500
N670 G01 X34.8017 Y-1.1628 Z24.9995 C0.0000 B0.0000 F1500
N680 G01 X34.8016 Y1.1632 Z24.9995 C0.0000 B0.0000 F1500
N690 G01 X34.8016 Y3.4892 Z24.9995 C0.0000 B0.0000 F1500
N700 G01 X34.8016 Y5.8152 Z24.9995 C0.0000 B0.0000 F1500
N710 G01 X34.8016 Y8.1412 Z24.9995 C0.0000 B0.0000 F1500
N720 G01 X34.8016 Y10.4672 Z24.9995 C0.0000 B0.0000 F1500
N730 G01 X34.8016 Y12.7931 Z24.9995 C0.0000 B0.0000 F1500
N740 G01 X34.8016 Y15.1191 Z24.9995 C0.0000 B0.0000 F1500
N750 G01 X34.8016 Y17.4451 Z24.9995 C0.0000 B0.0000 F1500
N760 G01 X34.8015 Y19.7711 Z24.9995 C0.0000 B0.0000 F1500
N770 G01 X34.8015 Y22.0971 Z24.9995 C0.0000 B0.0000 F1500
N780 G01 X34.8015 Y24.4231 Z24.9995 C0.0000 B0.0000 F1500
N790 G01 X32.8021 Y25.5774 Z24.9995 C0.0000 B0.0000 F1500
N800 G01 X30.8026 Y26.7318 Z24.9995 C0.0000 B0.0000 F1500
N810 G01 X28.8031 Y27.8862 Z24.9995 C0.0000 B0.0000 F1500
N820 G01 X26.8037 Y29.0405 Z24.9995 C0.0000 B0.0000 F1500
N830 G01 X24.8042 Y30.1949 Z24.9995 C0.0000 B0.0000 F1500
N840 G01 X22.8048 Y31.3493 Z24.9995 C0.0000 B0.0000 F1500
N850 G01 X20.8053 Y32.5036 Z24.9995 C0.0000 B0.0000 F1500
N860 G01 X18.8058 Y33.6580 Z24.9995 C0.0000 B0.0000 F1500
N870 G01 X16.8064 Y34.8124 Z24.9995 C0.0000 B0.0000 F1500
N880 G01 X14.8069 Y35.9667 Z24.9995 C0.0000 B0.0000 F1500
N890 G01 X12.8075 Y37.1211 Z24.9995 C0.0000 B0.0000 F1500
N900 G01 X10.8080 Y38.2754 Z24.9995 C0.0000 B0.0000 F1500
N910 G01 X8.8085 Y39.4298 Z24.9995 C0.0000 B0.0000 F1500
N920 G01 X6.8091 Y40.5842 Z24.9995 C0.0000 B0.0000 F1500
```

```
N930 G01 X4.8096 Y41.7385 Z24.9995 C0.0000 B0.0000 F1500
N940 G01 X2.8102 Y42.8929 Z24.9995 C0.0000 B0.0000 F1500
N950 G01 X0.8107 Y44.0473 Z24.9995 C0.0000 B0.0000 F1500
N960 G01 X-1.1888 Y45.2016 Z24.9995 C0.0000 B0.0000 F1500
N970 G01 X-3.1882 Y46.3560 Z24.9995 C0.0000 B0.0000 F1500
N980 G01 X-5.1877 Y47.5103 Z24.9995 C0.0000 B0.0000 F1500
N990 G01 X-7.1871 Y48.6647 Z24.9995 C0.0000 B0.0000 F1500
N1000 G01 X-8.4804 Y46.7181 Z24.9995 C0.0000 B0.0000 F1500
N1010 G01 X-9.7736 Y44.7716 Z24.9995 C0.0000 B0.0000 F1500
N1020 G01 X-11.0668 Y42.8250 Z24.9995 C0.0000 B0.0000 F1500
N1030 G01 X-12.3601 Y40.8784 Z24.9995 C0.0000 B0.0000 F1500
N1040 G01 X-13.6533 Y38.9318 Z24.9995 C0.0000 B0.0000 F1500
N1050 G01 X-14.9465 Y36.9852 Z24.9995 C0.0000 B0.0000 F1500
N1060 G01 X-16.2397 Y35.0387 Z24.9995 C0.0000 B0.0000 F1500
N1070 G01 X-17.5330 Y33.0921 Z24.9995 C0.0000 B0.0000 F1500
N1080 G01 X-18.8262 Y31.1455 Z24.9995 C0.0000 B0.0000 F1500
N1090 G01 X-20.1194 Y29.1989 Z24.9995 C0.0000 B0.0000 F1500
N1100 G01 X-21.4126 Y27.2524 Z24.9995 C0.0000 B0.0000 F1500
N1110 G01 X-22.7059 Y25.3058 Z24.9995 C0.0000 B0.0000 F1500
N1120 G01 X-23.9991 Y23.3592 Z24.9995 C0.0000 B0.0000 F1500
N1130 G01 X-25.2923 Y21.4126 Z24.9995 C0.0000 B0.0000 F1500
N1140 G01 X-26.5855 Y19.4660 Z24.9995 C0.0000 B0.0000 F1500
N1150 G01 X-27.8788 Y17.5195 Z24.9995 C0.0000 B0.0000 F1500
N1160 G01 X-29.1720 Y15.5729 Z24.9995 C0.0000 B0.0000 F1500
N1170 G01 X-30.4652 Y13.6263 Z24.9995 C0.0000 B0.0000 F1500
N1180 G01 X-31.7585 Y11.6797 Z24.9995 C0.0000 B0.0000 F1500
N1190 G01 X-33.0517 Y9.7332 Z24.9995 C0.0000 B0.0000 F1500
N1200 G01 X-34.3449 Y7.7866 Z24.9995 C0.0000 B0.0000 F1500
N1210 G01 X-35.6381 Y5.8400 Z24.9995 C0.0000 B0.0000 F1500
N1220 G01 X-36.9314 Y3.8934 Z24.9995 C0.0000 B0.0000 F1500
N1230 G01 X-38.2246 Y1.9469 Z24.9995 C0.0000 B0.0000 F1500
N1240 G01 X-39.5178 Y0.0003 Z24.9995 C0.0000 B0.0000 F1500
N1250 G01 X-39.5178 Y0.0003 Z26.9995 C0.0000 B0.0000 F3000
N1260 G01 X-39.5178 Y0.0003 Z28.9995 C0.0000 B0.0000 F3000
N1270 G01 X-39.5178 Y0.0003 Z30.9995 C0.0000 B0.0000 F3000
N1280 G01 X-39.5178 Y0.0003 Z32.9995 C0.0000 B0.0000 F3000
N1290 G01 X-39.5178 Y0.0003 Z34.9995 C0.0000 B0.0000 F3000
N1300 G00 X-39.5178 Y0.0003 Z36.8746 C0.0000 B0.0000 F10000
N1310 G00 X-39.5178 Y0.0003 Z38.7496 C0.0000 B0.0000 F10000
N1320 G00 X-39.5178 Y0.0003 Z40.6247 C0.0000 B0.0000 F10000
N1330 G00 X-39.5178 Y0.0003 Z42.4998 C0.0000 B0.0000 F10000
N1340 G00 X-39.5178 Y0.0003 Z44.3748 C0.0000 B0.0000 F10000
N1350 G00 X-39.5178 Y0.0003 Z46.2499 C0.0000 B0.0000 F10000
N1360 G00 X-39.5178 Y0.0003 Z48.1249 C0.0000 B0.0000 F10000
N1370 G00 X-39.5178 Y0.0003 Z50.0000 C0.0000 B0.0000 F10000
N1380 G00 X-38.0171 Y0.0001 Z50.0000 C0.0000 B0.0000 F10000
N1390 G00 X-36.5164 Y0.0000 Z50.0000 C0.0000 B0.0000 F10000
N1400 G00 X-35.0157 Y-0.0001 Z50.0000 C0.0000 B0.0000 F10000
N1410 G00 X-33.5150 Y-0.0003 Z50.0000 C0.0000 B0.0000 F10000
```

```
N1420 G00 X-33.5150 Y-0.0003 Z48.1249 C0.0000 B0.0000 F10000
N1430 G00 X-33.5150 Y-0.0003 Z46.2499 C0.0000 B0.0000 F10000
N1440 G00 X-33.5150 Y-0.0003 Z44.3748 C0.0000 B0.0000 F10000
N1450 G00 X-33.5150 Y-0.0003 Z42.4998 C0.0000 B0.0000 F10000
N1460 G00 X-33.5150 Y-0.0003 Z40.6247 C0.0000 B0.0000 F10000
N1470 G00 X-33.5150 Y-0.0003 Z38.7496 C0.0000 B0.0000 F10000
N1480 G00 X-33.5150 Y-0.0003 Z36.8746 C0.0000 B0.0000 F10000
N1490 G00 X-33.5150 Y-0.0003 Z34.9995 C0.0000 B0.0000 F10000
N1500 G01 X-33.5150 Y-0.0003 Z32.9995 C0.0000 B0.0000 F900
N1510 G01 X-33.5150 Y-0.0003 Z30.9995 C0.0000 B0.0000 F900
N1520 G01 X-33.5150 Y-0.0003 Z28.9995 C0.0000 B0.0000 F900
N1530 G01 X-33.5150 Y-0.0003 Z26.9995 C0.0000 B0.0000 F900
N1540 G01 X-33.5150 Y-0.0003 Z24.9995 C0.0000 B0.0000 F900
N1550 G01 X-32.2470 Y-1.9088 Z24.9995 C0.0000 B0.0000 F1500
N1560 G01 X-30.9791 Y-3.8173 Z24.9995 C0.0000 B0.0000 F1500
N1570 G01 X-29.7111 Y-5.7257 Z24.9995 C0.0000 B0.0000 F1500
N1580 G01 X-28.4432 Y-7.6342 Z24.9995 C0.0000 B0.0000 F1500
N1590 G01 X-27.1752 Y-9.5427 Z24.9995 C0.0000 B0.0000 F1500
N1600 G01 X-25.9073 Y-11.4512 Z24.9995 C0.0000 B0.0000 F1500
N1610 G01 X-24.6393 Y-13.3597 Z24.9995 C0.0000 B0.0000 F1500
N1620 G01 X-23.3714 Y-15.2682 Z24.9995 C0.0000 B0.0000 F1500
N1630 G01 X-22.1034 Y-17.1767 Z24.9995 C0.0000 B0.0000 F1500
N1640 G01 X-20.8355 Y-19.0852 Z24.9995 C0.0000 B0.0000 F1500
N1650 G01 X-19.5675 Y-20.9936 Z24.9995 C0.0000 B0.0000 F1500
N1660 G01 X-18.2996 Y-22.9021 Z24.9995 C0.0000 B0.0000 F1500
N1670 G01 X-17.0316 Y-24.8106 Z24.9995 C0.0000 B0.0000 F1500
N1680 G01 X-15.7637 Y-26.7191 Z24.9995 C0.0000 B0.0000 F1500
N1690 G01 X-14.4957 Y-28.6276 Z24.9995 C0.0000 B0.0000 F1500
N1700 G01 X-13.2278 Y-30.5361 Z24.9995 C0.0000 B0.0000 F1500
N1710 G01 X-11.9598 Y-32.4446 Z24.9995 C0.0000 B0.0000 F1500
N1720 G01 X-10.6919 Y-34.3531 Z24.9995 C0.0000 B0.0000 F1500
N1730 G01 X-9.4239 Y-36.2616 Z24.9995 C0.0000 B0.0000 F1500
N1740 G01 X-8.1560 Y-38.1700 Z24.9995 C0.0000 B0.0000 F1500
N1750 G01 X-6.8881 Y-40.0785 Z24.9995 C0.0000 B0.0000 F1500
N1760 G01 X-5.6201 Y-41.9870 Z24.9995 C0.0000 B0.0000 F1500
N1770 G01 X-3.6522 Y-40.8508 Z24.9995 C0.0000 B0.0000 F1500
N1780 G01 X-1.6843 Y-39.7147 Z24.9995 C0.0000 B0.0000 F1500
N1790 G01 X0.2835 Y-38.5785 Z24.9995 C0.0000 B0.0000 F1500
N1800 G01 X2.2514 Y-37.4423 Z24.9995 C0.0000 B0.0000 F1500
N1810 G01 X4.2193 Y-36.3062 Z24.9995 C0.0000 B0.0000 F1500
N1820 G01 X6.1872 Y-35.1700 Z24.9995 C0.0000 B0.0000 F1500
N1830 G01 X8.1551 Y-34.0338 Z24.9995 C0.0000 B0.0000 F1500
N1840 G01 X10.1230 Y-32.8976 Z24.9995 C0.0000 B0.0000 F1500
N1850 G01 X12.0908 Y-31.7615 Z24.9995 C0.0000 B0.0000 F1500
N1860 G01 X14.0587 Y-30.6253 Z24.9995 C0.0000 B0.0000 F1500
N1870 G01 X16.0266 Y-29.4891 Z24.9995 C0.0000 B0.0000 F1500
N1880 G01 X17.9945 Y-28.3529 Z24.9995 C0.0000 B0.0000 F1500
N1890 G01 X19.9624 Y-27.2168 Z24.9995 C0.0000 B0.0000 F1500
N1900 G01 X21.9302 Y-26.0806 Z24.9995 C0.0000 B0.0000 F1500
```

```
N1910 G01 X23.8981 Y-24.9444 Z24.9995 C0.0000 B0.0000 F1500
N1920 G01 X25.8660 Y-23.8082 Z24.9995 C0.0000 B0.0000 F1500
N1930 G01 X27.8339 Y-22.6721 Z24.9995 C0.0000 B0.0000 F1500
N1940 G01 X29.8018 Y-21.5359 Z24.9995 C0.0000 B0.0000 F1500
N1950 G01 X29.8018 Y-19.2689 Z24.9995 C0.0000 B0.0000 F1500
N1960 G01 X29.8017 Y-17.0020 Z24.9995 C0.0000 B0.0000 F1500
N1970 G01 X29.8017 Y-14.7350 Z24.9995 C0.0000 B0.0000 F1500
N1980 G01 X29.8017 Y-12.4681 Z24.9995 C0.0000 B0.0000 F1500
N1990 G01 X29.8017 Y-10.2011 Z24.9995 C0.0000 B0.0000 F1500
N2000 G01 X29.8017 Y-7.9341 Z24.9995 C0.0000 B0.0000 F1500
N2010 G01 X29.8017 Y-5.6672 Z24.9995 C0.0000 B0.0000 F1500
N2020 G01 X29.8017 Y-3.4002 Z24.9995 C0.0000 B0.0000 F1500
N2030 G01 X29.8017 Y-1.1333 Z24.9995 C0.0000 B0.0000 F1500
N2040 G01 X29.8016 Y1.1337 Z24.9995 C0.0000 B0.0000 F1500
N2050 G01 X29.8016 Y3.4007 Z24.9995 C0.0000 B0.0000 F1500
N2060 G01 X29.8016 Y5.6676 Z24.9995 C0.0000 B0.0000 F1500
N2070 G01 X29.8016 Y7.9346 Z24.9995 C0.0000 B0.0000 F1500
N2080 G01 X29.8016 Y10.2015 Z24.9995 C0.0000 B0.0000 F1500
N2090 G01 X29.8016 Y12.4685 Z24.9995 C0.0000 B0.0000 F1500
N2100 G01 X29.8016 Y14.7355 Z24.9995 C0.0000 B0.0000 F1500
N2110 G01 X29.8015 Y17.0024 Z24.9995 C0.0000 B0.0000 F1500
N2120 G01 X29.8015 Y19.2694 Z24.9995 C0.0000 B0.0000 F1500
N2130 G01 X29.8015 Y21.5363 Z24.9995 C0.0000 B0.0000 F1500
N2140 G01 X27.8336 Y22.6725 Z24.9995 C0.0000 B0.0000 F1500
N2150 G01 X25.8657 Y23.8086 Z24.9995 C0.0000 B0.0000 F1500
N2160 G01 X23.8978 Y24.9448 Z24.9995 C0.0000 B0.0000 F1500
N2170 G01 X21.9299 Y26.0809 Z24.9995 C0.0000 B0.0000 F1500
N2180 G01 X19.9620 Y27.2170 Z24.9995 C0.0000 B0.0000 F1500
N2190 G01 X17.9941 Y28.3532 Z24.9995 C0.0000 B0.0000 F1500
N2200 G01 X16.0262 Y29.4893 Z24.9995 C0.0000 B0.0000 F1500
N2210 G01 X14.0583 Y30.6255 Z24.9995 C0.0000 B0.0000 F1500
N2220 G01 X12.0904 Y31.7616 Z24.9995 C0.0000 B0.0000 F1500
N2230 G01 X10.1225 Y32.8977 Z24.9995 C0.0000 B0.0000 F1500
N2240 G01 X8.1546 Y34.0339 Z24.9995 C0.0000 B0.0000 F1500
N2250 G01 X6.1867 Y35.1700 Z24.9995 C0.0000 B0.0000 F1500
N2260 G01 X4.2188 Y36.3062 Z24.9995 C0.0000 B0.0000 F1500
N2270 G01 X2.2508 Y37.4423 Z24.9995 C0.0000 B0.0000 F1500
N2280 G01 X0.2829 Y38.5784 Z24.9995 C0.0000 B0.0000 F1500
N2290 G01 X-1.6850 Y39.7146 Z24.9995 C0.0000 B0.0000 F1500
N2300 G01 X-3.6529 Y40.8507 Z24.9995 C0.0000 B0.0000 F1500
N2310 G01 X-5.6208 Y41.9869 Z24.9995 C0.0000 B0.0000 F1500
N2320 G01 X-6.8887 Y40.0784 Z24.9995 C0.0000 B0.0000 F1500
N2330 G01 X-8.1566 Y38.1699 Z24.9995 C0.0000 B0.0000 F1500
N2340 G01 X-9.4245 Y36.2614 Z24.9995 C0.0000 B0.0000 F1500
N2350 G01 X-10.6924 Y34.3529 Z24.9995 C0.0000 B0.0000 F1500
N2360 G01 X-11.9604 Y32.4445 Z24.9995 C0.0000 B0.0000 F1500
N2370 G01 X-13.2283 Y30.5360 Z24.9995 C0.0000 B0.0000 F1500
N2380 G01 X-14.4962 Y28.6275 Z24.9995 C0.0000 B0.0000 F1500
N2390 G01 X-15.7641 Y26.7190 Z24.9995 C0.0000 B0.0000 F1500
```

```
N2400 G01 X-17.0320 Y24.8105 Z24.9995 C0.0000 B0.0000 F1500
N2410 G01 X-18.2999 Y22.9020 Z24.9995 C0.0000 B0.0000 F1500
N2420 G01 X-19.5679 Y20.9936 Z24.9995 C0.0000 B0.0000 F1500
N2430 G01 X-20.8358 Y19.0851 Z24.9995 C0.0000 B0.0000 F1500
N2440 G01 X-22.1037 Y17.1766 Z24.9995 C0.0000 B0.0000 F1500
N2450 G01 X-23.3716 Y15.2681 Z24.9995 C0.0000 B0.0000 F1500
N2460 G01 X-24.6395 Y13.3596 Z24.9995 C0.0000 B0.0000 F1500
N2470 G01 X-25.9075 Y11.4512 Z24.9995 C0.0000 B0.0000 F1500
N2480 G01 X-27.1754 Y9.5427 Z24.9995 C0.0000 B0.0000 F1500
N2490 G01 X-28.4433 Y7.6342 Z24.9995 C0.0000 B0.0000 F1500
N2500 G01 X-29.7112 Y5.7257 Z24.9995 C0.0000 B0.0000 F1500
N2510 G01 X-30.9791 Y3.8172 Z24.9995 C0.0000 B0.0000 F1500
N2520 G01 X-32.2470 Y1.9088 Z24.9995 C0.0000 B0.0000 F1500
N2530 G01 X-33.5150 Y0.0003 Z24.9995 C0.0000 B0.0000 F1500
N2540 G01 X-33.5150 Y0.0003 Z26.9995 C0.0000 B0.0000 F3000
N2550 G01 X-33.5150 Y0.0003 Z28.9995 C0.0000 B0.0000 F3000
N2560 G01 X-33.5150 Y0.0003 Z30.9995 C0.0000 B0.0000 F3000
N2570 G01 X-33.5150 Y0.0003 Z32.9995 C0.0000 B0.0000 F3000
N2580 G01 X-33.5150 Y0.0003 Z34.9995 C0.0000 B0.0000 F3000
N2590 G00 X-33.5150 Y0.0003 Z36.8746 C0.0000 B0.0000 F10000
N2600 G00 X-33.5150 Y0.0003 Z38.7496 C0.0000 B0.0000 F10000
N2610 G00 X-33.5150 Y0.0003 Z40.6247 C0.0000 B0.0000 F10000
N2620 G00 X-33.5150 Y0.0003 Z42.4998 C0.0000 B0.0000 F10000
N2630 G00 X-33.5150 Y0.0003 Z44.3748 C0.0000 B0.0000 F10000
N2640 G00 X-33.5150 Y0.0003 Z46.2499 C0.0000 B0.0000 F10000
N2650 G00 X-33.5150 Y0.0003 Z48.1249 C0.0000 B0.0000 F10000
N2660 G00 X-33.5150 Y0.0003 Z50.0000 C0.0000 B0.0000 F10000
N2670 G00 X-32.0142 Y0.0001 Z50.0000 C0.0000 B0.0000 F10000
N2680 G00 X-30.5135 Y0.0000 Z50.0000 C0.0000 B0.0000 F10000
N2690 G00 X-29.0128 Y-0.0001 Z50.0000 C0.0000 B0.0000 F10000
N2700 G00 X-27.5121 Y-0.0003 Z50.0000 C0.0000 B0.0000 F10000
N2710 G00 X-27.5121 Y-0.0003 Z48.1249 C0.0000 B0.0000 F10000
N2720 G00 X-27.5121 Y-0.0003 Z46.2499 C0.0000 B0.0000 F10000
N2730 G00 X-27.5121 Y-0.0003 Z44.3748 C0.0000 B0.0000 F10000
N2740 G00 X-27.5121 Y-0.0003 Z42.4998 C0.0000 B0.0000 F10000
N2750 G00 X-27.5121 Y-0.0003 Z40.6247 C0.0000 B0.0000 F10000
N2760 G00 X-27.5121 Y-0.0003 Z38.7496 C0.0000 B0.0000 F10000
N2770 G00 X-27.5121 Y-0.0003 Z36.8746 C0.0000 B0.0000 F10000
N2780 G00 X-27.5121 Y-0.0003 Z34.9995 C0.0000 B0.0000 F10000
N2790 G01 X-27.5121 Y-0.0003 Z33.3328 C0.0000 B0.0000 F900
N2800 G01 X-27.5121 Y-0.0003 Z31.6662 C0.0000 B0.0000 F900
N2810 G01 X-27.5121 Y-0.0003 Z29.9995 C0.0000 B0.0000 F900
N2820 G01 X-27.5121 Y-0.0003 Z28.3328 C0.0000 B0.0000 F900
N2830 G01 X-27.5121 Y-0.0003 Z26.6662 C0.0000 B0.0000 F900
N2840 G01 X-27.5121 Y-0.0003 Z24.9995 C0.0000 B0.0000 F900
N2850 G01 X-26.2089 Y-1.9619 Z24.9995 C0.0000 B0.0000 F1500
N2860 G01 X-24.9056 Y-3.9235 Z24.9995 C0.0000 B0.0000 F1500
N2870 G01 X-23.6024 Y-5.8851 Z24.9995 C0.0000 B0.0000 F1500
N2880 G01 X-22.2991 Y-7.8467 Z24.9995 C0.0000 B0.0000 F1500
```



```
N2890 G01 X-20.9959 Y-9.8083 Z24.9995 C0.0000 B0.0000 F1500
N2900 G01 X-19.6926 Y-11.7699 Z24.9995 C0.0000 B0.0000 F1500
N2910 G01 X-18.3894 Y-13.7315 Z24.9995 C0.0000 B0.0000 F1500
N2920 G01 X-17.0862 Y-15.6931 Z24.9995 C0.0000 B0.0000 F1500
N2930 G01 X-15.7829 Y-17.6547 Z24.9995 C0.0000 B0.0000 F1500
N2940 G01 X-14.4797 Y-19.6163 Z24.9995 C0.0000 B0.0000 F1500
N2950 G01 X-13.1764 Y-21.5779 Z24.9995 C0.0000 B0.0000 F1500
N2960 G01 X-11.8732 Y-23.5395 Z24.9995 C0.0000 B0.0000 F1500
N2970 G01 X-10.5699 Y-25.5011 Z24.9995 C0.0000 B0.0000 F1500
N2980 G01 X-9.2667 Y-27.4627 Z24.9995 C0.0000 B0.0000 F1500
N2990 G01 X-7.9635 Y-29.4244 Z24.9995 C0.0000 B0.0000 F1500
N3000 G01 X-6.6602 Y-31.3860 Z24.9995 C0.0000 B0.0000 F1500
N3010 G01 X-5.3570 Y-33.3476 Z24.9995 C0.0000 B0.0000 F1500
N3020 G01 X-4.0537 Y-35.3092 Z24.9995 C0.0000 B0.0000 F1500
N3030 G01 X-2.1300 Y-34.1985 Z24.9995 C0.0000 B0.0000 F1500
N3040 G01 X-0.2063 Y-33.0878 Z24.9995 C0.0000 B0.0000 F1500
N3050 G01 X1.7174 Y-31.9772 Z24.9995 C0.0000 B0.0000 F1500
N3060 G01 X3.6411 Y-30.8665 Z24.9995 C0.0000 B0.0000 F1500
N3070 G01 X5.5648 Y-29.7558 Z24.9995 C0.0000 B0.0000 F1500
N3080 G01 X7.4885 Y-28.6452 Z24.9995 C0.0000 B0.0000 F1500
N3090 G01 X9.4122 Y-27.5345 Z24.9995 C0.0000 B0.0000 F1500
N3100 G01 X11.3359 Y-26.4238 Z24.9995 C0.0000 B0.0000 F1500
N3110 G01 X13.2596 Y-25.3132 Z24.9995 C0.0000 B0.0000 F1500
N3120 G01 X15.1833 Y-24.2025 Z24.9995 C0.0000 B0.0000 F1500
N3130 G01 X17.1070 Y-23.0918 Z24.9995 C0.0000 B0.0000 F1500
N3140 G01 X19.0307 Y-21.9812 Z24.9995 C0.0000 B0.0000 F1500
N3150 G01 X20.9544 Y-20.8705 Z24.9995 C0.0000 B0.0000 F1500
N3160 G01 X22.8781 Y-19.7598 Z24.9995 C0.0000 B0.0000 F1500
N3170 G01 X24.8018 Y-18.6491 Z24.9995 C0.0000 B0.0000 F1500
N3180 G01 X24.8018 Y-16.3180 Z24.9995 C0.0000 B0.0000 F1500
N3190 G01 X24.8017 Y-13.9868 Z24.9995 C0.0000 B0.0000 F1500
N3200 G01 X24.8017 Y-11.6556 Z24.9995 C0.0000 B0.0000 F1500
N3210 G01 X24.8017 Y-9.3245 Z24.9995 C0.0000 B0.0000 F1500
N3220 G01 X24.8017 Y-6.9933 Z24.9995 C0.0000 B0.0000 F1500
N3230 G01 X24.8017 Y-4.6621 Z24.9995 C0.0000 B0.0000 F1500
N3240 G01 X24.8017 Y-2.3310 Z24.9995 C0.0000 B0.0000 F1500
N3250 G01 X24.8016 Y0.0002 Z24.9995 C0.0000 B0.0000 F1500
N3260 G01 X24.8016 Y2.3314 Z24.9995 C0.0000 B0.0000 F1500
N3270 G01 X24.8016 Y4.6626 Z24.9995 C0.0000 B0.0000 F1500
N3280 G01 X24.8016 Y6.9937 Z24.9995 C0.0000 B0.0000 F1500
N3290 G01 X24.8016 Y9.3249 Z24.9995 C0.0000 B0.0000 F1500
N3300 G01 X24.8016 Y11.6561 Z24.9995 C0.0000 B0.0000 F1500
N3310 G01 X24.8016 Y13.9872 Z24.9995 C0.0000 B0.0000 F1500
N3320 G01 X24.8015 Y16.3184 Z24.9995 C0.0000 B0.0000 F1500
N3330 G01 X24.8015 Y18.6496 Z24.9995 C0.0000 B0.0000 F1500
N3340 G01 X22.8778 Y19.7602 Z24.9995 C0.0000 B0.0000 F1500
N3350 G01 X20.9541 Y20.8708 Z24.9995 C0.0000 B0.0000 F1500
N3360 G01 X19.0303 Y21.9815 Z24.9995 C0.0000 B0.0000 F1500
N3370 G01 X17.1066 Y23.0921 Z24.9995 C0.0000 B0.0000 F1500
```



```
N3380 G01 X15.1829 Y24.2027 Z24.9995 C0.0000 B0.0000 F1500
N3390 G01 X13.2592 Y25.3134 Z24.9995 C0.0000 B0.0000 F1500
N3400 G01 X11.3354 Y26.4240 Z24.9995 C0.0000 B0.0000 F1500
N3410 G01 X9.4117 Y27.5346 Z24.9995 C0.0000 B0.0000 F1500
N3420 G01 X7.4880 Y28.6452 Z24.9995 C0.0000 B0.0000 F1500
N3430 G01 X5.5642 Y29.7559 Z24.9995 C0.0000 B0.0000 F1500
N3440 G01 X3.6405 Y30.8665 Z24.9995 C0.0000 B0.0000 F1500
N3450 G01 X1.7168 Y31.9771 Z24.9995 C0.0000 B0.0000 F1500
N3460 G01 X-0.2069 Y33.0878 Z24.9995 C0.0000 B0.0000 F1500
N3470 G01 X-2.1307 Y34.1984 Z24.9995 C0.0000 B0.0000 F1500
N3480 G01 X-4.0544 Y35.3090 Z24.9995 C0.0000 B0.0000 F1500
N3490 G01 X-5.3576 Y33.3474 Z24.9995 C0.0000 B0.0000 F1500
N3500 G01 X-6.6608 Y31.3858 Z24.9995 C0.0000 B0.0000 F1500
N3510 G01 X-7.9640 Y29.4242 Z24.9995 C0.0000 B0.0000 F1500
N3520 G01 X-9.2672 Y27.4626 Z24.9995 C0.0000 B0.0000 F1500
N3530 G01 X-10.5704 Y25.5010 Z24.9995 C0.0000 B0.0000 F1500
N3540 G01 X-11.8736 Y23.5394 Z24.9995 C0.0000 B0.0000 F1500
N3550 G01 X-13.1768 Y21.5778 Z24.9995 C0.0000 B0.0000 F1500
N3560 G01 X-14.4800 Y19.6162 Z24.9995 C0.0000 B0.0000 F1500
N3570 G01 X-15.7832 Y17.6546 Z24.9995 C0.0000 B0.0000 F1500
N3580 G01 X-17.0865 Y15.6930 Z24.9995 C0.0000 B0.0000 F1500
N3590 G01 X-18.3897 Y13.7315 Z24.9995 C0.0000 B0.0000 F1500
N3600 G01 X-19.6929 Y11.7699 Z24.9995 C0.0000 B0.0000 F1500
N3610 G01 X-20.9961 Y9.8083 Z24.9995 C0.0000 B0.0000 F1500
N3620 G01 X-22.2993 Y7.8467 Z24.9995 C0.0000 B0.0000 F1500
N3630 G01 X-23.6025 Y5.8851 Z24.9995 C0.0000 B0.0000 F1500
N3640 G01 X-24.9057 Y3.9235 Z24.9995 C0.0000 B0.0000 F1500
N3650 G01 X-26.2089 Y1.9619 Z24.9995 C0.0000 B0.0000 F1500
N3660 G01 X-27.5121 Y0.0003 Z24.9995 C0.0000 B0.0000 F1500
N3670 M9
N3680 M30
```

## Index

interfaces (*module*), 78–113

- interfaces.get\_analog\_in (*function*), 79
- interfaces.get\_analog\_out (*function*), 79
- interfaces.get\_configurable\_digital\_in (*function*), 79
- interfaces.get\_configurable\_digital\_out (*function*), 80
- interfaces.get\_digital\_in (*function*), 80
- interfaces.get\_digital\_out (*function*), 81
- interfaces.get\_flag (*function*), 81
- interfaces.get\_standard\_analog\_in (*function*), 82
- interfaces.get\_standard\_analog\_out (*function*), 82
- interfaces.get\_standard\_digital\_in (*function*), 82
- interfaces.get\_standard\_digital\_out (*function*), 83
- interfaces.get\_tool\_analog\_in (*function*), 83
- interfaces.get\_tool\_digital\_in (*function*), 84
- interfaces.get\_tool\_digital\_out (*function*), 84
- interfaces.get\_tool\_digital\_output\_mode (*function*), 85
- interfaces.get\_tool\_output\_mode (*function*), 85
- interfaces.modbus\_add\_signal (*function*), 85
- interfaces.modbus\_delete\_signal (*function*), 86
- interfaces.modbus\_get\_signal\_status (*function*), 87
- interfaces.modbus\_send\_custom\_command (*function*), 87
- interfaces.modbus\_set\_digital\_input\_action (*function*), 88
- interfaces.modbus\_set\_output\_register (*function*), 89
- interfaces.modbus\_set\_output\_signal (*function*), 89
- interfaces.modbus\_set\_signal\_update\_frequency (*function*), 90
- interfaces.read\_input\_boolean\_register (*function*), 90
- interfaces.read\_input\_float\_register (*function*), 91
- interfaces.read\_input\_integer\_register (*function*), 91
- interfaces.read\_output\_boolean\_register (*function*), 92
- interfaces.read\_output\_float\_register (*function*), 92
- interfaces.read\_output\_integer\_register (*function*), 93
- interfaces.read\_port\_bit (*function*), 93
- interfaces.read\_port\_register (*function*), 94
- interfaces.rpc\_factory (*function*), 94
- interfaces.rtde\_set\_watchdog (*function*), 95
- interfaces.set\_analog\_inputrange (*function*), 96
- interfaces.set\_analog\_out (*function*), 97
- interfaces.set\_configurable\_digital\_out (*function*), 97
- interfaces.set\_digital\_out (*function*), 98
- interfaces.set\_flag (*function*), 98
- interfaces.set\_standard\_analog\_out (*function*), 98
- interfaces.set\_standard\_digital\_out (*function*), 99
- interfaces.set\_tool\_communication (*function*), 99
- interfaces.set\_tool\_digital\_out (*function*), 100
- interfaces.set\_tool\_digital\_output\_mode (*function*), 101
- interfaces.set\_tool\_output\_mode (*function*), 101
- interfaces.set\_tool\_voltage (*function*), 101

interfaces.socket\_close (*function*), 102  
 interfaces.socket\_get\_var (*function*), 102  
 interfaces.socket\_open (*function*), 102  
 interfaces.socket\_read\_ascii\_float (*function*), 103  
 interfaces.socket\_read\_binary\_integer (*function*), 104  
 interfaces.socket\_read\_byte\_list (*function*), 105  
 interfaces.socket\_read\_line (*function*), 106  
 interfaces.socket\_read\_string (*function*), 107  
 interfaces.socket\_send\_byte (*function*), 108  
 interfaces.socket\_send\_int (*function*), 109  
 interfaces.socket\_send\_line (*function*), 109  
 interfaces.socket\_send\_string (*function*), 110  
 interfaces.socket\_set\_var (*function*), 110  
 interfaces.write\_output\_boolean\_register (*function*), 111  
 interfaces.write\_output\_float\_register (*function*), 111  
 interfaces.write\_output\_integer\_register (*function*), 112  
 interfaces.write\_port\_bit (*function*), 112  
 interfaces.write\_port\_register (*function*), 113  
 interfaces.zero\_ftsensor (*function*), 113  
 internals (*module*), 42–62  
   internals.force (*function*), 43  
   internals.get\_actual\_joint\_positions (*function*), 43  
   internals.get\_actual\_joint\_positions\_history (*function*), 43  
   internals.get\_actual\_joint\_speeds (*function*), 43  
   internals.get\_actual\_tcp\_pose (*function*), 44  
   internals.get\_actual\_tcp\_speed (*function*), 44  
   internals.get\_actual\_tool\_flange\_pose (*function*), 44  
   internals.get\_controller\_temp (*function*), 44  
   internals.get\_forward\_kin (*function*), 45  
   internals.get\_inverse\_kin (*function*), 45  
   internals.get\_joint\_temp (*function*), 46  
   internals.get\_joint\_torques (*function*), 46  
   internals.get\_steptime (*function*), 47  
   internals.get\_target\_joint\_positions (*function*), 47  
   internals.get\_target\_joint\_speeds (*function*), 47  
   internals.get\_target\_payload (*function*), 47  
   internals.get\_target\_payload\_cog (*function*), 48  
   internals.get\_target\_tcp\_pose (*function*), 48  
   internals.get\_target\_tcp\_speed (*function*), 48  
   internals.get\_target\_waypoint (*function*), 48  
   internals.get\_tcp\_force (*function*), 49  
   internals.get\_tcp\_offset (*function*), 49  
   internals.get\_tool\_accelerometer\_reading (*function*), 49  
   internals.get\_tool\_current (*function*), 49  
   internals.is\_steady (*function*), 50  
   internals.is\_within\_safety\_limits (*function*), 50  
   internals.popup (*function*), 50  
   internals.powerdown (*function*), 51  
   internals.set\_gravity (*function*), 51

- internals.set\_payload (*function*), 51
- internals.set\_payload\_cog (*function*), 52
- internals.set\_payload\_mass (*function*), 52
- internals.set\_tcp (*function*), 53
- internals.sleep (*function*), 53
- internals.str\_at (*function*), 53
- internals.str\_cat (*function*), 54
- internals.str\_empty (*function*), 55
- internals.str\_find (*function*), 56
- internals.str\_len (*function*), 56
- internals.str\_sub (*function*), 57
- internals.sync (*function*), 58
- internals.textmsg (*function*), 58
- internals.to\_num (*function*), 59
- internals.to\_str (*function*), 60
- internals.tool\_contact (*function*), 61
- internals.tool\_contact\_examples (*function*), 61
- ioconfiguration (*module*), 113–121
  - ioconfiguration.modbus\_set\_runstate\_dependent\_choice (*function*), 114
  - ioconfiguration.set\_analog\_outputdomain (*function*), 114
  - ioconfiguration.set\_configurable\_digital\_input\_action (*function*), 114
  - ioconfiguration.set\_gp\_boolean\_input\_action (*function*), 115
  - ioconfiguration.set\_input\_actions\_to\_default (*function*), 116
  - ioconfiguration.set\_runstate\_configurable\_digital\_output\_to\_value (*function*), 116
  - ioconfiguration.set\_runstate\_gp\_boolean\_output\_to\_value (*function*), 117
  - ioconfiguration.set\_runstate\_standard\_analog\_output\_to\_value (*function*), 117
  - ioconfiguration.set\_runstate\_standard\_digital\_output\_to\_value (*function*), 118
  - ioconfiguration.set\_runstate\_tool\_digital\_output\_to\_value (*function*), 118
  - ioconfiguration.set\_standard\_analog\_input\_domain (*function*), 119
  - ioconfiguration.set\_standard\_digital\_input\_action (*function*), 119
  - ioconfiguration.set\_tool\_analog\_input\_domain (*function*), 120
  - ioconfiguration.set\_tool\_digital\_input\_action (*function*), 120
- motion (*module*), 14–42
  - motion.conveyor\_pulse\_decode (*function*), 15
  - motion.encoder\_enable\_pulse\_decode (*function*), 15
  - motion.encoder\_enable\_set\_tick\_count (*function*), 16
  - motion.encoder\_get\_tick\_count (*function*), 17
  - motion.encoder\_set\_tick\_count (*function*), 17
  - motion.encoder\_unwind\_delta\_tick\_count (*function*), 18
  - motion.end\_force\_mode (*function*), 19
  - motion.end\_freedrive\_mode (*function*), 20
  - motion.end\_screw\_driving (*function*), 20
  - motion.end\_teach\_mode (*function*), 20
  - motion.force\_mode (*function*), 20
  - motion.force\_mode\_example (*function*), 21
  - motion.force\_mode\_set\_damping (*function*), 22
  - motion.force\_mode\_set\_gain\_scaling (*function*), 22
  - motion.freedrive\_mode (*function*), 23

motion.get\_conveyor\_tick\_count (*function*), 23  
 motion.get\_target\_tcp\_pose\_along\_path (*function*), 23  
 motion.get\_target\_tcp\_speed\_along\_path (*function*), 23  
 motion.movec (*function*), 24  
 motion.movej (*function*), 25  
 motion.movel (*function*), 26  
 motion.movep (*function*), 27  
 motion.path\_offset\_disable (*function*), 28  
 motion.path\_offset\_enable (*function*), 28  
 motion.path\_offset\_get (*function*), 29  
 motion.path\_offset\_set (*function*), 29  
 motion.path\_offset\_set\_alpha\_filter (*function*), 30  
 motion.path\_offset\_set\_max\_offset (*function*), 31  
 motion.position\_deviation\_warning (*function*), 31  
 motion.reset\_revolution\_counter (*function*), 32  
 motion.screw\_driving (*function*), 33  
 motion.servoc (*function*), 34  
 motion.servoj (*function*), 35  
 motion.set\_conveyor\_tick\_count (*function*), 36  
 motion.set\_pos (*function*), 37  
 motion.set\_safety\_mode\_transition\_hardness (*function*), 38  
 motion.speedj (*function*), 38  
 motion.speedl (*function*), 38  
 motion.stop\_conveyor\_tracking (*function*), 39  
 motion.stopj (*function*), 39  
 motion.stopl (*function*), 40  
 motion.teach\_mode (*function*), 40  
 motion.track\_conveyor\_circular (*function*), 40  
 motion.track\_conveyor\_linear (*function*), 41

processpath (*module*), 121–127  
   processpath.mc\_add\_circular (*function*), 122  
   processpath.mc\_add\_linear (*function*), 122  
   processpath.mc\_add\_path (*function*), 123  
   processpath.mc\_get\_target\_rtcp\_speed (*function*), 124  
   processpath.mc\_initialize (*function*), 124  
   processpath.mc\_load\_path (*function*), 125  
   processpath.mc\_run\_motion (*function*), 126  
   processpath.mc\_set\_pcs (*function*), 126  
   processpath.mc\_set\_speed\_factor (*function*), 127

urmath (*module*), 62–78  
   urmath.acos (*function*), 63  
   urmath.asin (*function*), 63  
   urmath.atan (*function*), 63  
   urmath.atan2 (*function*), 64  
   urmath.binary\_list\_to\_integer (*function*), 64  
   urmath.ceil (*function*), 65  
   urmath.cos (*function*), 65  
   urmath.d2r (*function*), 66

urmath.floor (*function*), 66  
urmath.get\_list\_length (*function*), 66  
urmath.integer\_to\_binary\_list (*function*), 67  
urmath.interpolate\_pose (*function*), 67  
urmath.length (*function*), 68  
urmath.log (*function*), 68  
urmath.norm (*function*), 69  
urmath.normalize (*function*), 69  
urmath.point\_dist (*function*), 70  
urmath.pose\_add (*function*), 70  
urmath.pose\_dist (*function*), 71  
urmath.pose\_inv (*function*), 71  
urmath.pose\_sub (*function*), 72  
urmath.pose\_trans (*function*), 72  
urmath.pow (*function*), 73  
urmath.r2d (*function*), 74  
urmath.random (*function*), 74  
urmath.rotvec2rpy (*function*), 74  
urmath.rpy2rotvec (*function*), 75  
urmath.sin (*function*), 76  
urmath.sqrt (*function*), 76  
urmath.tan (*function*), 77  
urmath.wrench\_trans (*function*), 77