# Autocontrol Scheduler Documentation

This code implements a Flask server that acts as an interface between a web application and a device API that controls scientific instruments. The server listens for POST requests with task information and puts the task into a priority queue. A background task continuously executes tasks from the queue in order of their priority until the server is stopped.

## General Approach

The Autocontrol server is a device-agnostic scheduler that passes tasks to devices using a priority queue and transfers samples between devices. It is aware of the occupational state of devices and the location of samples, which allows it to time task execution. Autocontrol attempts to run samples in parallel on multi-channeled hardware and finds routes between devices based on a simple rule set.

Each device is channel-based, even if it has only a single channel. Autocontrol runs tasks either on manually set channels, or it determines the channel itself. For the latter, there are currently three channel modes implemented on a per-device basis:

1. None: Any of the free channels is used.
2. Reuse: Any of the free channels is used. If a sample used a particular channel in the past this channel will be reused.
3. New: Each time, a new channel will be used.

When consistently auto-selecting channels, the algorithm will determine a route of the sample material through connected devices, which is reused for all successive tasks concerning this sample and device if channel mode is set to ‘reuse’. If channels are available and the device is not busy, successive tasks are processed in parallel. Such tasks can be for different samples. A sample is defined by having the same sample number, otherwise, Autocontrol is agnostic. Samples with a lower sample number have a higher priority in task execution. Tasks that are submitted earlier have a higher priority than later tasks.

Each device needs to implement its set of tasks from a list of tasks specified below that Autocontrol can handle. When executing a task on a device, Autocontrol expects the device to mark the respective channel it acts on as BUSY. When the task is completed, the device needs to mark the channel as IDLE. This indicates the operational status of the channel, which is independent of the physical occupancy of a channel with material. Autocontrol keeps track of the physical occupancy of a channel based on the successful execution of a task.

Measurement task attach data to the task that was read from the device. It is currently stored together with the task in a SQLite database.

## Overview

The server has three endpoints:

1. '/': the default endpoint that indicates the server has started successfully;
2. '/put': a POST request endpoint that submits a task to the device queue. The request data is of type autocontrol.task.Task and must include the following fields:

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Options** |
| id | UUID | task ID |  |
| tasks | dict | a dictionary describing the task to be executed by the instrument. This field is passed on to the instrument API |  |
| task\_type | str | a generic label for the type of a task as recognized by Autocontrol | init, prepare, transfer, measure, no\_channel, shut down |

The tasks subfield contains a list of sub-tasks farmed out to different instrument associated with the task\_type given. Often, only one task for a particular device is provided, but for example, transfer tasks might contain a list of several devices involved in the transfer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Options / comments** |
| id | UUID | Sub-task ID |  |
| method\_data | dict | json-like object, usually dict, that contains the methods that are submitted to the device for execution |  |
| device | str | device name for the task |  |
| channel | int | the channel to be used, None for auto-select |  |
| target\_device | str | name of the device the material is transferred to | Only for transfer tasks |
| target\_channel | int | channel on the target device to be used, auto-select if None. | Only for transfer tasks |
| md | dict | metadata that will be saved with the results |  |

Optional parameters can be provided for every sub-task.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Task type** | **Description** | **Options** |
| acquisition\_time | float | measure | Data acquisition time in seconds | If none given, the measurement needs to terminate on its own. |
| channel\_mode | None or str | init | Overwrites device-level channel mode setting (see below) | None (default), ‘reuse’, ‘new’ |
| device\_address | str | init | HTML or other address to communicate with device |  |
| device\_type | str | init | The device type for selecting the correct API | lh, qcmd |
| force | Bool | transfer | Determines if a transfer is executed even if target channel is already occupied | False (default), True |
| non\_channel\_source | str | transfer | A source that is not a channel such as a reservoir for rinses. |  |
| non\_channel\_target | str | transfer | A target that is not a channel such as a waste line. |  |
| simulated | Bool | init | Sets a device to simulated (test) or not | True, False (default) |
| wait\_for\_queue\_to\_empty | Bool | shutdown | Waits for all scheduled and active tasks to finish before shutting down the server | False (default), True |

1. '/shutdown': a POST request endpoint that stops the Flask server. The request data may include the 'wait\_for\_queue\_to\_empty' field, which, if True, waits for all tasks in the queue to be executed before shutting down the server.

## Task Implementation Details

### init

Instruments need to be initialized using the ‘init’ task type before any other task. The ‘tasks’ subfield may contain entries with additional keys listed below (see above for details on individual arguments). As currently implemented, initialization tasks take a higher priority over all other tasks.

* channel: default = 1, number of channels in the device
* channel\_mode: None (default), ‘reuse’, ‘new’. Defines how channels are assigned to tasks with the same sample number: no particular algorithm, reuse previously used channels, always use a new channel.
* device\_address: the URL for HTTP requests or any other implemented communication method.
* device\_type: string that describes the type of device (‘lh’, ‘qcmd’)
* simulated: Boolean value that sets the API mode to a simulated device if True
* …: any additional data fields to be passed on to the device itself

### prepare

Prepare a sample in a device, which is an action that ends with placing material into a channel. Device-specific recipes shut be placed under the ‘task’ key. Required keys are:

* channel: default = None, channel to put the prepared material in

### transfer

Transfers material from one device to another following the rules set by ‘channel\_mode’ for each device. Arguments are:

* channel: default = None, source channel
* device: source device
* target\_channel: default = None, target channel
* target\_device: target device

under ‘task’

* force: default = False, disable checks on target channel occupation

### measure

Initiates a measurement on the device. Arguments are:

* channel: default = None, channel in which the measurement is done

under ‘task’:

* acquisition\_time: default = None, measurement time in seconds

### no\_channel

This is a channel-less task that will set the entire device to BUSY while being executed, and back to UP when finished.

### shutdown

To stop the server gracefully, shutdown\_server is called, it waits for the background task thread to exit. If the wait\_for\_queue\_to\_empty parameter is set to True, the function waits for the queue to empty before actually shutting down the server. Arguments are:

under ‘task’:

* wait\_for\_queue\_to\_empty: default = False, wait for priority queue and active tasks

## Example Usage

For an example usage see the integration test.

## Device API Requirements

The device API expects the following methods for each device, if applicable.

**communicate**

Communicates with device and return response.

:param command: HTTP POST request command field

:param value: HTTP POST request value field

:return: status, response (autocontrol status), (str or None) response from POST request

**execute\_task**

Routes tasks to the appropriate subroutines

:param task: task to execute

:return status: autocontrol status

**get\_channel\_status**

Retrieves the status of a channel.

:param channel: (int) default=0, the channel to be used.

:return status: (str) IDLE, BUSY

**get\_device\_status**

Retrieves the status of a device independent from its channels. This means while any particular channel can be BUSY, the device itself can be UP and ready to receive commands for other channels or, for example, an initialization command. Accepting new commands can paused using the BUSY or DOWN statuses.

:return status: (autocontrol status) UP, BUSY, DOWN, INVALID, ERROR

**read**

Retrieves the measurement data of a channel collected since the measurement was started.

:param channel: (int) default=0, the channel to be used.

:return status, data: (autocontrol status, json) status and dict-like data

**init**

Performs an initialization of the device. During initialization, the device status is expected to be set to DOWN. After successful initialization, the device should return UP for the device status.

:param task: (task\_container) task object

:return status: (autocontrol status)

**measure**

Submits a measurement task to a device. It is similar to the ‘prepare’ function and ensures that measurement data is read out after the task is completed.

:param task: (task\_container) task object

:return status: (str) status concerning the task submission

**prepare**

Submits a preparation task to the device, such as sample mixing, or a pre-measurement activity.

:param task: (task\_container) task object

:return status: (autocontrol status)

**transfer**

Submits a transfer of material request from one device to another.

:param task: (task\_container) task object

:return status: (autocontrol status)

**shut\_down**

Performs a shutdown of the device

:return status: (autocontrol status)

## Database visualization

The server starts a Streamlit application to display submitted, active, and previous tasks.