# 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

Server endpoints:

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **POST data** | **GET data** | **Description** |
| / |  |  | the default endpoint that indicates the server has started successfully |
| /cancel | ‘task\_id’: the ID of the task to be cancelled |  | Cancels a tasks from the autocontrol priority queue |
| /pause |  |  | Pauses the scheduling queue |
| /put | A serialized autocontrol task object as detailed below (task.json()) | Dict with keys: ‘response’: response message, ‘task\_id’: task id | submits a task to the autocontrol queue |
| /queue\_inspect |  | Dict with keys ‘task\_*i*’ of the *i*=0…*n*-1 tasks. | Retrieves a serialized copy of all task in the priority queue |
| /reset |  |  | Deletes all tasks from the queue and clears the channel occupancy of every device. It removes all sample number and ID information from autocontrol. |
| /restart |  |  | Same as /reset with additionally deleting all device initializations. |
| /resume |  |  | Resumes the scheduling queue after pausing. |
| /shutdown | Optional bool, ‘wait\_for\_queue\_to\_empty’: if True server waits until all tasks in the queue have been executed. |  | stops the Flask server |

## Task Implementation

### Task Structure

The task and subtask structures are defined in task\_struct.py. The task parent structure contains the following fields.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field** | **Type** | **Mandatory for submission** | **Description** | **Options** |
| dependency\_id | UUID | no | If not None, the task will not be executed before the task with the dependency ID has started executing |  |
| dependency\_ sample\_number | int | no | If not None, the task will not be executed before the task with the dependency sample number has started executing |  |
| id | UUID | no | task ID, auto-assigned when not provided |  |
| md | dict | no | Any metadata associated with the task |  |
| priority | float | no | An execution priority associated with the task. If not provided it will be auto-generated based on sample number and submission time. |  |
| sample\_id | UUID | no | A sample ID. If not provided, it will be auto-generated. |  |
| sample\_number | int | no | A sample number. Tasks that share a sample number typically share the same path through the device network, for example, preparations and transfers ending up in the same measurement device where they subsequently modify a sample. This can involve multiple injections of material (sub-samples). If a new sample ID is provided but no sample number, a new increasing sample number will be generated. I neither ID nor sample number is provided, the sample number defaults to 1. |  |
| tasks | list | yes | A list of dictionaries describing the task to be executed by the instrument. This field is passed on to the instrument API |  |
| task\_history | list | no | List of UUIDS of previous tasks that handled the material associated with this task |  |
| task\_type | str | yes | A string 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** | **Mandatory at submission** | **Description** | **Options / comments** |
| id | UUID | no | Sub-task ID |  |
| device | str | yes | device name for the subtask |  |
| channel | int | no | the channel to be used, None for auto-select |  |
| method\_data | dict | yes | json-like object, usually dict, that contains the methods that are submitted to the device for execution. An empty dict can be submitted to the device. |  |
| md | dict |  | metadata associated with the subtask, including 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, injection |
| force | bool | transfer | Determines if a transfer is executed even if target channel is already occupied | False (default), True |
| non\_channel\_storage | str | transfer | A source or target 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. |  |
| sample\_mixing | bool | init | Defines whether a device can process tasks out of order by sample number. |  |
| 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 |

### Task Type Specifics

#### init

Instruments must 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 are prioritized 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: a 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 each device's rules as set by ‘channel\_mode.’ Transfer tasks should have a list of subtasks under the tasks field of task. Each subtask relates to one device in the transfer chain of two or more devices.

* channel: default = None, source channel
* device: source device

Some devices are declared ‘passive’ in their internal initialization. Those do not have an active transfer of material in or out. A passive device can be between two active devices in a three-device transfer or at the end of a transfer. In the latter, the previous contents of the device are expected to be drained to waste.

#### 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.