# Bluesky Flask Server Documentation

This code implements a Flask server that acts as an interface between a web application and a Bluesky 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.

## 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 must include the following fields:

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Options** |
| task | 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 affecting prioritization | init, prepare, transfer, measure, shut down, exit |
| sample\_number | int | an ascending sample ID |  |
| device | str | device name for the task |  |
| channel | int | the channel to be used, None for auto-select |  |
| md | dict | metadata that will be saved with the results |  |

For transfer tasks, additionally the following data fields are required:

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Options** |
| target\_device | str | name of the device the materialed is transferred to |  |
| target\_channel | int | channel on the target device to be used, auto-select if None. |  |

Optional parameters can be provided in the ‘task’ data field.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Description** | **Options** |
| acquisition\_time | float | Data acquisition time in seconds |  |
| channel\_mode | None or str | Overwrites device-level channel mode setting (see below) | None, ‘reuse’, ‘new’ |

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.

## Implementation Details

The server uses Flask version 1.1.2 to set up the endpoints. A background task handles the device task queue. The function background\_task is executed as a separate thread that continuously takes one task from the queue and executes it until the server is stopped. The method queue\_put puts tasks in the priority queue. The priority is determined by the sample number and submission time. The lower the sample number and the earlier the submission, the higher the priority.

Instruments need to be initialize using the ‘init’ task type. The ‘task’ subfield may contain the additional keys listed below. Repeated initialization overwrites previous ones.

* channel: default = 1, number of channels in the device
* channel\_mode: None, ‘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
* …: any additional data fields to be passed on to the device itself

Multiple parallel channels allow processing separate samples in parallel. Each task can either explicitly define a channel or, by setting the data field ‘channel’ to None, use an auto-selected channel. 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 channels are available and the device is not busy, successive samples are processed in parallel

To stop the server gracefully, shutdown\_server is called, which sets the app\_shutdown flag to True and 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.

## Example Usage

import requests

# send a PUT request to the server with the task data

url = "http://localhost:5003/put"

data = {

"sample": {field1: …, field2: …, },

"sample\_number": 1,

"measurement\_channel": None,

"md": {},

"task\_type": "measure",

"device": "QCMD"

}

response = requests.put(url, json=data)

# send a POST request to stop the server

url = "http://localhost:5003/shutdown"

data = {"wait\_for\_queue\_to\_empty": True}

response = requests.post(url, json=data)

## Bluesky API Requirements

Devices are dynamically initialized via the ‘init’ task. The Bluesky API expects the following methods for each device, if applicable.

**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: (str) 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: (str, json) status and dict-like data

**init**

Performs an initialization of the device

:param number\_of\_channels: (int) default=1, sets the numbers of channels to be used

:param \*\*kwargs: any additional device-specific settings

:return status, number\_of\_channels: (str, int) status and number of successfully initialized channels

**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: (json) dict-like object containing all task-relevant information

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

: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: (json) dict-like object containing all task-relevant information

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

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

**transfer**

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

:param task: (json) dict-like object containing all task-relevant information

:param channel: (int) default=0, the channel from which the material is transferred from.

:param target\_device: (str) The device name to which the material is transferred to.

:param target\_channel: (int) default=0, the channel to which the material is transferred to.

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

**shut\_down**

Performs a shutdown of the device

:return status: (str, int) status and number of successfully initialized channels

Generally, status strings are ‘success’ or contain an error message.

## Database visualization

We can imagine using a tool like Streamlit to display submitted, active, and previous tasks.