# 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 Bluesky queue. The request data must include the following fields:
   * task: a dictionary describing the task to be executed by the instrument. This field is passed on to the instrument API.
   * sample\_number: (integer) an ascending sample ID;
   * channel: (integer) the channel to be used in case parallel channels are supported, otherwise None
   * md: a dictionary containing metadata to be saved with the measurement data;
   * task\_type: a generic label that determines the type of a task affecting how they are prioritized. Options: 'init', 'prepare', 'transfer', 'measure', 'shut down', 'exit'
   * device: a string that identifies the device executing the task.
3. '/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 Bluesky 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 server uses the bluesky\_api package to control scientific instruments, and 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 key ‘number\_of\_channels’, setting up multiple channels for this device.

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)