# uREST

## Background

This library is designed to enable simple API’s to be built on micro-controllers, based on a sub-set of the REST API design principles, and inspired by the design of the [Apollo DSKY](https://history-computer.com/apollo-guidance-computer/) guidance computer. Rather than build a full HTTP server stack, including JSON parser, and supporting the full complexity of modern REST API’s, this library aims to support simple operations in a resource constrained environment.

Like the DSKY unit, it is assumed that all the ‘objects’ representing the states we are interested in are held in ‘[nouns](https://dlove24.github.io/urest/urest/api/base.html)’. The HTTP actions then represent ‘verbs’ which dictate the actions on the noun. So each API call is then in the form of ‘verb-noun’; e.g. ‘GET /led’, or ‘PUT /led’. Valid verb actions are

| Verb | HTTP Method | Action |
| --- | --- | --- |
| Get | GET | Return the current state of the requested noun. |
| Set | PUT | Set the requested noun to *exactly* the specified state. This is assumed to be idempotent, with the resultant state matching exactly the request from the client. |
| Update | POST | Update the state requested noun. This is *not* assumed to be idempotent: for instance asking a noun to move between two states on each update. |
| Delete | DELETE | Remove the current state of the noun, and return to a the default state. This does *not* remove the noun from the API: only the state currently held by the API. |

In all cases the body of the HTTP request is a simple collection of ‘key: value’ pairs, formatted as a [JSON](https://www.json.org/json-en.html) object. Only a sub-set of the JSON specification is used: in particular multiple objects are not allowed, and nor are arrays (i.e. ‘[]’) of any sort. This both simplifies the parsing, and especially the memory required for the parser, and reinforces the intent to support only minimal API’s.

## Installation

A package of this library is provided on PyPi as [urest-mp](https://pypi.org/project/urest-mp/). This can be installed with the normal Python tools, and should also install to boards runnning MicroPython under [Thonny](https://thonny.org/).

For manual installation, everything under the urest directory should be copied to the appropriate directory on the MicroPython board, usually /lib. The library can then be imported as normal, e.g.

python from urest.http import RESTServer from urest.api import APIBase

See the documentation for the [examples](https://dlove24.github.io/urest/urest/examples/index.html) for more detailed guidance on the use of the library. This code is also available in the urest/examples folder, or as the library urest.examples when the package is installed.

## Debugging

Console output from the urest.http.server.RESTServer is controlled by the standard \_\_debug\_\_ flag. By default no output will be sent to the ‘console’ *unless* the \_\_debug\_\_ flag is True.

**Note:** that in the standard Python environments the status of the \_\_debug\_\_ flag is often controlled by the optimisation level of the interpreter: see the standard [Python documentation](https://docs.python.org/3/using/cmdline.html#cmdoption-O) for more details. For MicroPython the status of the \_\_debug\_\_ flag is set by [internal constants](https://docs.micropython.org/en/latest/library/micropython.html#micropython.opt_level). However if the \_\_debug\_\_ constant is set whilst a programming is running the [results may be unexpected](https://forum.micropython.org/viewtopic.php?t=6839), due to optimisations undertaken by the MicroPython lexer. Instead for MicroPython set the status of the \_\_debug\_\_ flag in the platform standard boot.py or similar: see the documentation for the specific port for more details.

## Design

The core of the library is a simple HTTP server, specialised to the delivery of a REST-like API instead of a general HTTP server. The design, and the use of the asyncio library, is inspired by the [MicroPython HTTP Server](https://github.com/erikdelange/MicroPython-HTTP-Server) by Erik de Lange. This library uses a roughly similar structure for the core of the asyncio event loop, and especially in the design of the [RESTServer](https://dlove24.github.io/urest/urest/http/server.html) class.

Key differences include

* Support for PUT, POST and DELETE operations, in addition to GET. These are required for an API server, and also form the ‘verbs’ of the actions allowed on the ‘nouns’ by the API built on-top of this library.
* A more object-oriented design of the call/response handler, made easier this library is *not* a general HTTP server. For instance Python getters and setters are used where possible for input validation, and the central API response is based on the [APIBase](https://dlove24.github.io/urest/urest/api/base.html) abstract base class
* A more explicit validation of input from the network layer, especially in the assumption that all input is by default hostile. This library should serve as an example of best-practice in protocol handling; at least in the slightly resource constrained environment of MicroPython
* This implementation is principally a teaching library, so the [Documentation](https://dlove24.github.io/urest/urest) should be at least as important as the ‘code’. Where possible all algorithms and implementation techniques should also be explained as fully as possible, or at least linked to reference standards/implementations
* For consistency, all code should also be in the format standardised by the [Black](https://github.com/psf/black) library. This makes it easier to co-ordinate external code and documentation with the implementation documented here.

## Known Implementations

* Raspberry Pi Pico W (MicroPython 3.4)