UOS Interface

Release 0.0.0-alpha.1

Steve Richardson

Oct 17, 2022

CONTENTS:

Python Interface for remote communication and control of a UOS Compliant Microcontroller.

The uosinterface tooling source code is divided into two packages:

- *Hardware Abstraction* - Containing device definitions and low level abstraction which can be imported to introduce UOS automation directly on a client machine.
- *Webapp* - Containing a high level web server which can be used to provide network access to a uos controller via a dashboard or API requests.

CHAPTER

ONE

HARDWARE ABSTRACTION

The UOS interface hardware abstraction layer provides a common interface for control of UOS devices.

1.1 Supported Hardware

Supported devices are enumerated and defined in *uosinterface.hardware.devices.py*.

All devices are defined using the *Devices* abstraction class.

class uosinterface.hardware.uosabstractions.**Device**(*name: str*, *interfaces: list*, *functions_enabled: dict*, *digital_pins: dict = <factory>*, *analogue_pins: dict = <factory>*, *aux_params: dict = <factory>*)

Define an implemented UOS device dictionary.

get_compatible_pins(*function_name: str*) → {}

Returns a dict of pin objects that are suitable for a function.

Parameters
function_name – the string name of the UOS Schema function.

Returns
Dict of pin objects, keyed on pin index.

1.2 Abstraction Layer

Devices can be accessed through the hardware layer by instantiating a *UOSDevice*. By default the device is used in a lazy manner, where references to the interface opened and closed automatically as required for functions.

Example usage:

This is the *hello world* usage for turning on the arduino on-board pin 13 LED.

```
from uosinterface.hardware import UOSDevice
from uosinterface.hardware.devices import ARDUINO_NANO_3
from uosinterface.hardware.devices import Interface

device = UOSDevice(identity = ARDUINO_NANO_3, address = "/dev/ttyUSB0")
device.set_gpio_output(pin=13, level=1)  # switch on LED
```

Note: that individual pins and functions must be enabled and supported by the *Device*.

class uosinterface.hardware.__init__.**UOSDevice**(*identity: Union[str, Device]*, *address: str*, *interface: Interface = Interface.USB*, ***kwargs*)

Class for high level object-orientated control of UOS devices.

Variables

- **identity** – The type of device, this is must have a valid device in the config.
- **connection** – Compliant connection string for identifying the device and interface.
- **device** – Device definitions as parsed from a compatible ini.
- **__kwargs** – Connection specific / optional parameters.
- **__device_interface** – Lower level communication protocol layer.

close()

Releases connection, must be called explicitly if loading is eager.

Raises

UOSCommunicationError - Problem closing the connection to an active device.

device

alias of *Device*

get_adc_input(*pin: int*, *level: int*, *volatility: int = 0*) → ComResult

Reads the current 10 bit ADC value.

Parameters

- **pin** – The index of the analogue pin to read
- **level** – Reserved for future use.
- **volatility** – How volatile should the command be, use constants from HardwareCOM.

Returns

ComResult object containing the ADC readings.

get_gpio_config(*pin: int*, ***kwargs*) → ComResult

Reads the configuration for a digital pin on the device.

Parameters

- **pin** – Defines the pin for config querying.
- **kwargs** – Control arguments accepts volatility.

Returns

ComResult object containing the system information.

get_gpio_input(*pin: int*, *level: int*, *volatility: int = 0*) → ComResult

Reads a GPIO pins level from device and returns the value.

Parameters

- **pin** – The numeric number of the pin as defined in the dictionary for that device.
- **level** – Not used currently, future will define pull-up state.
- **volatility** – How volatile should the command be, use constants from HardwareCOM.

Returns

ComResult object.

get_system_info(***kwargs*) → ComResult

Reads the UOS version and device type.

Parameters
kwargs – Control arguments, accepts volatility.

Returns
ComResult object containing the system information.

hard_reset(***kwargs*) → ComResult

Hard reset functionality for the UOS Device.

is_lazy() → bool

Checks the loading type of the device lazy or eager.

Returns
Boolean, true is lazy.

open()

Connects to the device, explict calls are normally not required.

Raises
UOSCommunicationError - Problem opening a connection.

reset_all_io(***kwargs*) → ComResult

Executes the reset IO at the defined volatility level.

set_gpio_output(*pin: int*, *level: int*, *volatility: int = 0*) → ComResult

Sets a pin to digital output mode and sets a level on that pin.

Parameters

- **pin** – The numeric number of the pin as defined in the dictionary for that device.
- **level** – The output level, 0 - low, 1 - High.
- **volatility** – How volatile should the command be, use constants from HardwareCOM.

Returns
ComResult object.

1.3 Hardware Interfaces

- Stub
- USB Serial

CHAPTER

TWO

WEBAPP

The webapp package simultaneously serves a dashboard and an API for control of UOS hardware.

2.1 Dashboard

The UOS Interface can be configured and used entirely a graphical web-dashboard interface.

2.2 API Automation Layer

The UOS Interface supports automated control of UOS devices through a RESTful web API. This API interface exists on the url `http://served-address/api`.

The interface is designed to be replicate the functionality of the *Hardware Abstraction*. To achieve stateless operation all API operations are self-contained and stand-alone.

Example Usage:

The standard format is `http://served-address/api/version/instruction?device_arguments&instruction_arguments`

- *version* - Define the API version to use, certain UOS versions and API levels may not be compatible.
- *instruction* - Provides the name of the instruction being used from the hardware abstraction layer.
- *device_arguments* - Must provide address and identity at minimum.
- *instruction_arguments* - Must provide all non-optional arguments for the HAL instruction being used.

Note: Arguments can be provided in any order but must all be seperated using the URL & delimiter.

Example Usage:

This is the *hello world* usage for turning on the arduino on-board pin 13 LED.

```
http://served-address/api/1.0/set_gpio_output?address=/dev/ttyUSB0&identity=arduino_nano&pin=13&level=1
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX