IoT-UPnP Documentation

Release 0.1.dev10

Remi BONNET

Sep 11, 2018

Contents

This project is a little pure Python library to annonce a device by UPnP. It's made for IoT developers to let them show their device on network discovery in any OS.

With IoT-UPnP, you can show your device on all computers with Windows Explorer (network discovery). Users can doucle-click on the device to open the device configuration page. The device will be shown without any software installation on user's computers.

Developers can use UPnP to announce custom services and let others applications use these services. For example, if cameras announce a service "exemple-org:service:camera-image", you can made a software witch easy discover all cameras with this service.

CHAPTER 1

Requirement

You need Python 3.x to use this library.

CHAPTER 2

documentation

2.1 Quickstart

2.1.1 Usage with dict

For easy class initialisation, you can call the constructor with a dict.

```
import upnp

device = upnp.Device({
  'deviceType': 'urn:sadmin-fr:device:demo:1',
  'friendlyName': 'UPnP Test',
  'uuid': '00a56575-78fa-40fe-b107-8f4b5043a2b0',
  'manufacturer': 'BONNET',
  'manufacturerURL': 'http://sadmin.fr'
})

service = upnp.Service({
  'serviceType': 'sadmin-fr:service:dummy',
  'serviceId': 'sadmin-fr:serviceId:1',
})

device.addService(service)

server = upnp.Annoncer(device)
server.initLoop()
server.notify()
server.foreaver()
server.dispose()
```

Note: The UUID of the device must be unique and not change even if the application restart.

2.1.2 Usage with attribute

You can use attributes to manage objects.

```
import upnp

device = upnp.Device()
device.deviceType = 'urn:sadmin-fr:device:demo:1'
device.friendlyName = 'UPnP Test'
device.uuid = '00a56575-78fa-40fe-b107-8f4b5043a2b0'
device.manufacturer = 'BONNET'
device.manufacturerURL = 'http://sadmin.fr'

service = upnp.Service()
service.serviceType = 'sadmin-fr:service:dummy'
service.serviceId = 'sadmin-fr:serviceId:1'

device.addService(service)
server = upnp.Annoncer(device)
server.initLoop()
server.notify()
server.foreaver()
server.dispose()
```

Note: The UUID of the device must be unique and not change even if the application restart.

With attributes, you can also change device informations after the initialization.

2.1.3 asyncio and UPnP

IoT-UPnP use asyncio for events handling. You can specify the loop which will handle events with the initLoop method of the announcer class.

```
  loop = asyncio.get_event_loop()

  server = Annoncer(device)
  server.initLoop(loop)

  loop.run_forever()

The announcer class will use the ``asyncio.get_event_loop()`` when no loop
are specified.
```

IoT-UPnP require python 3.x. It use the following modules:

- asyncio: for the main event loop
- ssdp: base library for SSDP (a component of UPnP)
- netifaces: Network interfaces discovery (to retrieve IPs)

They are tree important objects:

class upnp.**Announcer**(*device*, *httpPort=5000*, *netBind='0.0.0.0'*)

Annoncer main class

See *upnp.UPnP.Announcer*

class upnp.**Device**(*obj=None*)
An UPnP device on the Network

See *upnp.Objects.Device*

class upnp.**Service**(*obj=None*)
A service on a device

See *upnp.Objects.Service*

All objects can be set with theirs attributes or by passing a dict on the contructor.

2.1.4 Goal

UPnP device have services that need to be announced. Services can be controlled and work as SOAP api. The class Announcer is used to anounce a device which have one or many services.

On UPnP references, a device can also have many embedded devices.

To make a UPnP device, you need to make a service, add this service to a device and them, announce this device as a root device.

2.2 upnp

2.2.1 upnp package

Submodules

upnp.HTTP module

class upnp.HTTP.**DescriptionAnswer**(*request*, *upnp*)
Bases: *upnp.HTTP.HttpAnswer*

HTTP success, describe a device (XML)

describeDevice(*device*)
Add a device description to the answer

Parameters **device** (upnp.Objects.Device) – Device to describe

describeIcon(*icon*)
Add an icon description to the answer

Parameters **icon** (upnp.Objects.Icon) – Icon to describe

describeService(*service*)
Add a service description to the answer

Parameters **service** (upnp.Objects.Service) – Service to describe

execute()
Prepare the description answer

class upnp.HTTP.**HTTP**(*annoncer*, *port*, *netbind*)
Bases: object

The main HTTP server class

dispose()
Close HTTP handling

initLoop(*loop*)
Add HTTP handlers on the asyncio loop

Parameters **loop** (*asyncio.AbstractEventLoop*) – Loop to use

class upnp.HTTP.**HttpAnswer**(*request*)
Bases: object

Class to construct an HTTP response

execute()
Need to be overrided by subclass. It's the execute process.

pprint()
Show the packet as human readable

write(*writer*)
Send the response

Parameters **writer** (*asyncio.StreamWriter*) – Writer to send pakcet

class upnp.HTTP.**HttpRequest**(*method*, *path*, *version*, *headers*)
Bases: object

HTTP Request informations

pprint()
Print human readable request

class upnp.HTTP.**HttpServer**(*config*)
Bases: object

class to handle asyncio events on HTTP service

HttpRouting(*request*)
A simple routing by path for incomming requests

Parameters **request** (upnp.HTTP.HttpRequest) – The incomming request

Returns The answer to execute

Return type *upnp.HTTP.HttpAnswer*

InConnection(*reader*, *writer*)
A new incomming connection

Parameters

- **reader** (*asyncio.StreamReader*) – Request input
- **writer** (*asyncio.StreamWriter*) – Stream to answer

class upnp.HTTP.**ScpdAnswer**(*request*, *upnp*)
Bases: *upnp.HTTP.HttpAnswer*

HTTP success, Describe a service API

execute()
Prepare the answer

class upnp.HTTP.**ServerErrorAnswer**(*request*)
Bases: *upnp.HTTP.HttpAnswer*

HTTP error response

execute()
Prepare the response

upnp.Objects module

class upnp.Objects.**Device**(*obj=None*)
Bases: upnp.Objects._BaseObj

An UPnP device on the Network

addDevice(*device*)
Add an embedded device

Parameters **device** (upnp.Device) – The embedded device to add

addService(*service*)
Add a service on this device

Parameters **service** (upnp.Service) – The service to add

class upnp.Objects.**Icon**(*obj=None*)
Bases: upnp.Objects._BaseObj

An device icon (don't work on Windows)

class upnp.Objects.**Service**(*obj=None*)
Bases: upnp.Objects._BaseObj

A service on a device

upnp.SSDP module

class upnp.SSDP.**AnnouncerService**
Bases: ssdp.SimpleServiceDiscoveryProtocol

Endpoint for UDP packets (used by asyncio)

connection_made(*transport*)
Called when the connection is made

Parameters **transport** (*asyncio.BaseTransport*) – The endpoint transport

request_received(*request*, *addr*)
Handle a new SSDP packet

Parameters

- **request** – Request informations
- **addr** – Adress of the peer

response_received(*response*, *addr*)
Not used

class upnp.SSDP.**Answer**(*config*, *status_code*, *reason*)
Bases: ssdp.SSDPResponse

Answer packet for M-SEARCH queries

send(*device*, *ip*, *addr*)
Send the UDP answer

Parameters

- **device** (upnp.Objects.Device) – Device to announce
- **ip** (*str*) – IP of device
- **addr** (*str*) – Destination IP

sendto(*transport*, *addr*)
Rewriting of raw sending method (error at the end of headers).

Parameters

- **transport** (*asyncio.BaseTransport*) – Transport to use
- **addr** (*str*) – Destination address

class upnp.SSDP.**Notify**(*config*, *device*)
Bases: ssdp.SSDPRequest

SSDP Notify packet

send(*ip*, *usn=None*, *transport=None*)
Build and send the packet

Parameters

- **ip** (*str*) – Destination IP
- **usn** (*str*) – USN to notify
- **transport** (*asyncio.BaseTransport*) – Transport to use

sendto(*transport*, *addr*)
Rewriting of raw sending method (error at the end of headers).

Parameters

- **transport** (*asyncio.BaseTransport*) – Transport to use
- **addr** (*str*) – Destination address

class upnp.SSDP.**SSDP**(*annoncer*, *netBind='0.0.0.0'*)
Bases: object

Public class to handle SSDP protocol

dispose()
Close SSDP handling

initLoop(*loop*)
Initiate an asyncio event loop

Parameters **loop** (*asyncio.AbstractEventLoop*) – An asyncio event loop

notify()
Send NOTIFY packets

class upnp.SSDP.**SSDP_Protocol**(*config*)
Bases: object

Simple class to send packets

answer(*st*, *addr*)
Answer to an M-SEARCH query

Parameters

- **st** (*str*) – Queried subject
- **addr** (*(str, int)*) – Destination address of answer

getDevices(*st*)
Get all devices which match ST

Parameters **st** (*str*) – Queried subject to match

Returns List of devices that match query

Return type list(*upnp.Objects.Device*)

notify(*device*, *ip*)
Send NOTIFY packets for a device (services and embedded devices)

Parameters

- **device** (upnp.Objects.Device) – Device to announce
- **ip** (*str*) – IP of the device

provides(*usn*)
Check if USN is provided by root device

Parameters **usn** (*str*) – USN to test

Returns True if USN is provided

Return type bool

upnp.UPnP module

class upnp.UPnP.**Annoncer**(*device*, *httpPort=5000*, *netBind='0.0.0.0'*)
Bases: *upnp.UPnP.Announcer*

class upnp.UPnP.**Announcer**(*device*, *httpPort=5000*, *netBind='0.0.0.0'*)
Bases: object

Annoncer main class

bye()
Send BYE SSP packets on the network

dispose()
Clear loop

foreaver()
Run loop forever (test)

initLoop(*loop=None*)
Initialise an asyncio loop to handle network packages

Parameters **loop** (*asyncio.AbstractEventLoop*) – An asyncio event loop to initialise

notify()
Send NOTIFY SSDP packets on the network to announce a new device

Module contents

Module to announce a UPnP device on network

2.3 API Documentation

- genindex
- modindex
- search

Python Module Index

u

Index

U

W