

---

# **UPnPpy**

***Release 1.0.0***

**5kyc0d3r**

**Aug 14, 2020**



**TABLE OF CONTENTS:**

<b>1</b>	<b>What is UPnPy?</b>	<b>3</b>
1.1	Install . . . . .	3
<b>2</b>	<b>Examples:</b>	<b>13</b>
<b>3</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



Lightweight UPnP client library for Python.



## WHAT IS UPNPY?

UPnPPy is a lightweight UPnP client library for Python. It can discover devices and invoke UPnP actions.

### 1.1 Install

```
$ pip install upnpy
```

#### 1.1.1 Install

This guide shows various methods on how you can install UPnPPy.

##### From pip

The easiest way to install UPnPPy is to get it with pip:

```
$ pip install upnpy
```

##### From source

You can also clone the repository and install it from source:

```
$ git clone https://github.com/5kyc0d3r/upnpy.git && cd upnpy
$ python setup.py install
```

#### 1.1.2 Introduction

UPnPPy can discover UPnP devices, get a list of its services and invoke actions for the services. See below for the basic usage of UPnPPy.

##### Quickstart

Get the external IP address of an [Internet Gateway Device](#) :

```
import upnpy

upnp = upnpy.UPnP()

# Discover UPnP devices on the network
# Returns a list of devices e.g.: [Device <Broadcom ADSL Router>]
devices = upnp.discover()

# Select the IGD
# alternatively you can select the device directly from the list
# device = devices[0]
device = upnp.get_igd()

# Get the services available for this device
# Returns a list of services available for the device
# e.g.: [<Service (WANPPPPConnection) id="WANPPPPConnection.1">, ...]
device.get_services()

# We can now access a specific service on the device by its ID
# The IDs for the services in this case contain illegal values so we can't access it_
→by an attribute
# If the service ID didn't contain illegal values we could access it via an attribute_
→like this:
# service = device.WANPPPPConnection

# We will access it like a dictionary instead:
service = device['WANPPPPConnection.1']

# Get the actions available for the service
# Returns a list of actions for the service:
# [<Action name="SetConnectionType">,
#  <Action name="GetConnectionTypeInfo">,
#  <Action name="RequestConnection">,
#  <Action name="ForceTermination">,
#  <Action name="GetStatusInfo">,
#  <Action name="GetNATRSIPStatus">,
#  <Action name="GetGenericPortMappingEntry">,
#  <Action name="GetSpecificPortMappingEntry">,
#  <Action name="AddPortMapping">,
#  <Action name="DeletePortMapping">,
#  <Action name="GetExternalIPAddress">]
service.get_actions()

# Finally, get the external IP address
# Execute the action by its name
# Returns a dictionary: {'NewExternalIPAddress': 'xxx.xxx.xxx.xxx'}
service.GetExternalIPAddress()
```

#### Add a new port mapping to an Internet Gateway Device :

```
import upnpy

upnp = upnpy.UPnP()

# Discover UPnP devices on the network
# Returns a list of devices e.g.: [Device <Broadcom ADSL Router>]
devices = upnp.discover()
```

(continues on next page)



(continued from previous page)

```

# Select the IGD
# alternatively you can select the device directly from the list
# device = devices[0]
device = upnp.get_igd()

# Get the services available for this device
# Returns a list of services available for the device
# e.g.: [<Service (WANPPPPConnection) id="WANPPPPConnection.1">, ...]
device.get_services()

# We can now access a specific service on the device by its ID
# The IDs for the services in this case contain illegal values so we can't access it_
↳by an attribute
# If the service ID didn't contain illegal values we could access it via an attribute_
↳like this:
# service = device.WANPPPPConnection

# We will access it like a dictionary instead:
service = device['WANPPPPConnection.1']

# Get the actions available for the service
# Returns a list of actions for the service:
#   [<Action name="SetConnectionType">,
#    <Action name="GetConnectionTypeInfo">,
#    <Action name="RequestConnection">,
#    <Action name="ForceTermination">,
#    <Action name="GetStatusInfo">,
#    <Action name="GetNATRSIPStatus">,
#    <Action name="GetGenericPortMappingEntry">,
#    <Action name="GetSpecificPortMappingEntry">,
#    <Action name="AddPortMapping">,
#    <Action name="DeletePortMapping">,
#    <Action name="GetExternalIPAddress">]
service.get_actions()

# The action we are looking for is the "AddPortMapping" action
# Lets see what arguments the action accepts
# Use the get_input_arguments() or get_output_arguments() method of the action
# for a list of input / output arguments.
# Example output of the get_input_arguments method for the "AddPortMapping" action
# Returns a dictionary:
# [
#   {
#     "name": "NewRemoteHost",
#     "data_type": "string",
#     "allowed_value_list": []
#   },
#   {
#     "name": "NewExternalPort",
#     "data_type": "ui2",
#     "allowed_value_list": []
#   },
#   {
#     "name": "NewProtocol",
#     "data_type": "string",
#     "allowed_value_list": [

```

(continues on next page)

(continued from previous page)

```

#         "TCP",
#         "UDP"
#     ]
# },
# {
#     "name": "NewInternalPort",
#     "data_type": "ui2",
#     "allowed_value_list": []
# },
# {
#     "name": "NewInternalClient",
#     "data_type": "string",
#     "allowed_value_list": []
# },
# {
#     "name": "NewEnabled",
#     "data_type": "boolean",
#     "allowed_value_list": []
# },
# {
#     "name": "NewPortMappingDescription",
#     "data_type": "string",
#     "allowed_value_list": []
# },
# {
#     "name": "NewLeaseDuration",
#     "data_type": "ui4",
#     "allowed_value_list": []
# }
# ]
service.AddPortMapping.get_input_arguments()

# Finally, add the new port mapping to the IGD
# This specific action returns an empty dict: {}
service.AddPortMapping(
    NewRemoteHost='',
    NewExternalPort=80,
    NewProtocol='TCP',
    NewInternalPort=8000,
    NewInternalClient='192.168.1.3',
    NewEnabled=1,
    NewPortMappingDescription='Test port mapping entry from UPnPpy.',
    NewLeaseDuration=0
)

```

### 1.1.3 UPnPpy API

#### upnp package

#### Subpackages

#### upnp.py.soap package

## Submodules

### upnpy.soap.SOAP module

`upnpy.soap.SOAP.send(service, action, **action_arguments)`

#### Send a SOAP request

This function allows you to invoke an action for the target service.

#### Parameters

- **service** – DeviceService object
- **action** – SOAPAction object

**Returns** Request response data

- Example of a RAW SOAP request:

```
POST path control URL HTTP/1.0
HOST: hostname:portNumber
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
USER-AGENT: OS/version UPnP/1.1 product/version
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"

<?xml version="1.0"?>
<s:Envelope
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>in arg value</argumentName>
      <!-- other in args and their values go here, if any -->
    </u:actionName>
  </s:Body>
</s:Envelope>
```

## Module contents

### upnpy.ssdp package

## Submodules

### upnpy.ssdp.SSDPDevice module

**class** `upnpy.ssdp.SSDPDevice.SSDPDevice(address, response)`

Bases: `object`

#### Represents an SSDP device

Object for representing an SSDP device.

#### Parameters

- **address** (*tuple*) – SSDP device address

- **response** (*str*) – Device discovery response data

**class Service** (*service, service\_id, scpd\_url, control\_url, event\_sub\_url, base\_url*)

Bases: object

### Device service

Represents a service available on the device.

#### Parameters

- **service** (*str*) – Full service string (e.g.: `urn:schemas-upnp-org:service:WANIPConnection:1`)
- **service\_id** (*str*) – ID of the service
- **scpd\_url** (*str*) – SCPD URL of the service
- **control\_url** (*str*) – Control URL of the service
- **event\_sub\_url** (*str*) – Event Sub URL of the service
- **base\_url** (*str*) – Base URL of the service

**class Action** (*name, argument\_list, service*)

Bases: object

### Represents an action on a service

This class holds the details of a specific action available on a service.

#### Parameters

- **name** (*str*) – Name of the action
- **argument\_list** (*list*) – List of in / out arguments the action has
- **service** (`SSDPDevice.Service`) – The service to which this action belongs

**class Argument** (*name, direction, return\_value, related\_state\_variable*)

Bases: object

### Represents an argument for an action

This class holds the details of an argument for an action.

#### Parameters

- **name** (*str*) – Name of the argument
- **direction** (*str*) – Direction of the argument (in/out)
- **return\_value** (*str*) – Identifies at most one output argument as the return value
- **related\_state\_variable** – Defines the type of the argument

**get\_input\_arguments** ()

#### Get the input arguments for the action

Gets the input arguments for the action.

**Returns** List of input arguments for the action

**Return type** list

**get\_output\_arguments** ()

#### Get the output arguments for the action

Gets the output arguments for the action.

**Returns** List of output arguments for the action

**Return type** list

**class StateVariable** (*name, data\_type, allowed\_value\_list=None*)

Bases: object

**get\_actions** ()

**Return a list of actions available for the service**

Returns a list of available actions for the service.

**Returns** List of actions available for this service

**Return type** list

**get\_friendly\_name** ()

**Get the friendly name for the device**

Gets the device's friendly name

**Returns** Friendly name of the device

**Return type** str

**get\_services** ()

**Return a list of services available for the device**

Returns a list of available services for the device.

**Returns** List of services available for this device

**Return type** list

## upnpy.ssdp.SSDPHeader module

**class** upnpy.ssdp.SSDPHeader.SSDPHeader (\*\*headers)

Bases: object

**set\_header** (name, value)

**set\_headers** (\*\*headers)

**set\_method** (method)

## upnpy.ssdp.SSDPRequest module

**class** upnpy.ssdp.SSDPRequest.SSDPRequest (ssdp\_mcast\_addr='239.255.255.250',  
ssdp\_port=1900, \*\*headers)

Bases: *upnpy.ssdp.SSDPHeader.SSDPHeader*

**Create and perform an SSDP request**

**Parameters** **method** – SSDP request method [M-SEARCH or NOTIFY]

**m\_search** (discover\_delay=2, st='ssdp:all', \*\*headers)

**Perform an M-SEARCH SSDP request**

Send an SSDP M-SEARCH request for finding UPnP devices on the network.

**Parameters**

- **discover\_delay** (*int*) – Device discovery delay in seconds
- **st** (*str*) – Specify device Search Target
- **headers** (*str*) – Specify M-SEARCH specific headers

**Returns** List of device that replied

**Return type** list

**notify** (\*\*headers)

Perform a NOTIFY SSDP request

**Parameters** **headers** – Specify NOTIFY specific headers

**Returns**

## Module contents

### upnpy.upnp package

### Submodules

#### upnpy.upnp.UPnP module

**class** upnpy.upnp.UPnP.UPnP

Bases: object

#### UPnP object

A UPnP object used for device discovery

**discover** (*delay=2, \*\*headers*)

**Find UPnP devices on the network**

Find available UPnP devices on the network by sending an M-SEARCH request.

#### Parameters

- **delay** (*int*) – Discovery delay, amount of time in seconds to wait for a reply from devices
- **headers** – Optional headers for the request

**Returns** List of discovered devices

**Return type** list

**get\_igd** ()

**Get the Internet Gateway Device if available**

Gets the Internet Gateway device if it's available after discovery.

**Returns** The IGD if successful or raises a upnpy.exceptions.IGDError exception upon failure

**Return type** *SSDPDevice*

## Module contents

### Submodules

#### upnpy.utils module

upnpy.utils.**make\_http\_request** (*url, data=None, headers=None*)

**Helper function for making HTTP requests**

Helper function for making HTTP requests using urllib.

#### Parameters

- **url** (*str*) – The URL to which a request should be made

- **data** (*str*) – Provide data for the request. Request method will be set to POST if data is provided
- **headers** (*dict*) – Provide headers to send with the request

**Returns** A urllib.Request.urlopen object

**Return type** urllib.Request.urlopen

`upnpy.utils.parse_device_type(device_type)`

**Parse the the deviceType string**

Parses only the deviceType portion of the device type string

**Parameters** **device\_type** – Full device type string

**Returns** Parsed device type

**Return type** str

`upnpy.utils.parse_http_header(header, header_key)`

**Parse HTTP header value**

Parse the value of a specific header from a RAW HTTP response.

**Parameters**

- **header** (*str*) – String containing the RAW HTTP response and headers
- **header\_key** (*str*) – The header name of which to extract a value from

**Returns** The value of the header

**Return type** str

`upnpy.utils.parse_service_id(service_id)`

**Parse the the serviceID string**

Parses only the serviceID portion of the service ID string

**Parameters** **service\_id** – Full device type string

**Returns** Parsed service ID

**Return type** str

## upnpy.exceptions module

**exception** `upnpy.exceptions.ActionNotFoundError` (*message, action\_name*)

Bases: Exception

**Custom Action exception**

Custom Action exception class. Raised whenever a particular action is not available for a service.

**exception** `upnpy.exceptions.ArgumentError` (*message, argument*)

Bases: Exception

**Custom Argument exception**

Custom Argument exception class. Raised whenever an error has been detected during action invocation.

**exception** `upnpy.exceptions.IGDError`

Bases: Exception

**Custom Internet Gateway Device exception**

Custom IGD exception class. Raised whenever a problem with the IGD has been detected.

**exception** `upnpy.exceptions.NotAvailableError`

Bases: `Exception`

#### **Custom exception for when a certain URL could not be retrieved**

Custom element not retrieved exception class. Raised whenever a value needed to be accessed could not be retrieved from the URL.

**exception** `upnpy.exceptions.NotRetrievedError`

Bases: `Exception`

#### **Custom exception for objects that have not been retrieved**

Custom object not retrieved exception class. Raised whenever a certain property for a device or service was not retrieved.

**exception** `upnpy.exceptions.SOAPError` (*description, code*)

Bases: `Exception`

#### **Custom SOAP exception**

Custom SOAP exception class. Raised whenever an error response has been received during action invocation.

**exception** `upnpy.exceptions.ServiceNotFoundError` (*message, service\_name*)

Bases: `Exception`

#### **Custom Service exception**

Custom Service exception class. Raised whenever a particular service was not found for a device.

## **Module contents**

## **License**

MIT License

Copyright (c) 2019 5kyc0d3r

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



## EXAMPLES:

**Get the external IP address of an Internet Gateway Device :**

```
import upnpy

upnp = upnpy.UPnP()

# Discover UPnP devices on the network
# Returns a list of devices e.g.: [Device <Broadcom ADSL Router>]
devices = upnp.discover()

# Select the IGD
# alternatively you can select the device directly from the list
# device = devices[0]
device = upnp.get_igd()

# Get the services available for this device
# Returns a list of services available for the device
# e.g.: [<Service (WANPPPConnection) id="WANPPPConnection.1">, ...]
device.get_services()

# We can now access a specific service on the device by its ID
# The IDs for the services in this case contain illegal values so we can't access it
#   ↳ by an attribute
# If the service ID didn't contain illegal values we could access it via an attribute
#   ↳ like this:
# service = device.WANPPPConnection

# We will access it like a dictionary instead:
service = device['WANPPPConnection.1']

# Get the actions available for the service
# Returns a list of actions for the service:
#   [<Action name="SetConnectionType">,
#    <Action name="GetConnectionTypeInfo">,
#    <Action name="RequestConnection">,
#    <Action name="ForceTermination">,
#    <Action name="GetStatusInfo">,
#    <Action name="GetNATRSIPStatus">,
#    <Action name="GetGenericPortMappingEntry">,
#    <Action name="GetSpecificPortMappingEntry">,
#    <Action name="AddPortMapping">,
#    <Action name="DeletePortMapping">,
#    <Action name="GetExternalIPAddress">]
service.get_actions()
```

(continues on next page)

(continued from previous page)

```
# Finally, get the external IP address
# Execute the action by its name
# Returns a dictionary: {'NewExternalIPAddress': 'xxx.xxx.xxx.xxx'}
service.GetExternalIPAddress()
```

**Add a new port mapping to an Internet Gateway Device :**

```
import upnpy

upnp = upnpy.UPnP()

# Discover UPnP devices on the network
# Returns a list of devices e.g.: [Device <Broadcom ADSL Router>]
devices = upnp.discover()

# Select the IGD
# alternatively you can select the device directly from the list
# device = devices[0]
device = upnp.get_igd()

# Get the services available for this device
# Returns a list of services available for the device
# e.g.: [<Service (WANPPPPConnection) id="WANPPPPConnection.1">, ...]
device.get_services()

# We can now access a specific service on the device by its ID
# The IDs for the services in this case contain illegal values so we can't access it_
↳by an attribute
# If the service ID didn't contain illegal values we could access it via an attribute_
↳like this:
# service = device.WANPPPPConnection

# We will access it like a dictionary instead:
service = device['WANPPPPConnection.1']

# Get the actions available for the service
# Returns a list of actions for the service:
#   [<Action name="SetConnectionType">,
#    <Action name="GetConnectionTypeInfo">,
#    <Action name="RequestConnection">,
#    <Action name="ForceTermination">,
#    <Action name="GetStatusInfo">,
#    <Action name="GetNATRSIPStatus">,
#    <Action name="GetGenericPortMappingEntry">,
#    <Action name="GetSpecificPortMappingEntry">,
#    <Action name="AddPortMapping">,
#    <Action name="DeletePortMapping">,
#    <Action name="GetExternalIPAddress">]
service.get_actions()

# The action we are looking for is the "AddPortMapping" action
# Lets see what arguments the action accepts
# Use the get_input_arguments() or get_output_arguments() method of the action
# for a list of input / output arguments.
# Example output of the get_input_arguments method for the "AddPortMapping" action
# Returns a dictionary:
```

(continues on next page)

(continued from previous page)

```

# [
#     {
#         "name": "NewRemoteHost",
#         "data_type": "string",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewExternalPort",
#         "data_type": "ui2",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewProtocol",
#         "data_type": "string",
#         "allowed_value_list": [
#             "TCP",
#             "UDP"
#         ]
#     },
#     {
#         "name": "NewInternalPort",
#         "data_type": "ui2",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewInternalClient",
#         "data_type": "string",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewEnabled",
#         "data_type": "boolean",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewPortMappingDescription",
#         "data_type": "string",
#         "allowed_value_list": []
#     },
#     {
#         "name": "NewLeaseDuration",
#         "data_type": "ui4",
#         "allowed_value_list": []
#     }
# ]
service.AddPortMapping.get_input_arguments()

# Finally, add the new port mapping to the IGD
# This specific action returns an empty dict: {}
service.AddPortMapping(
    NewRemoteHost='',
    NewExternalPort=80,
    NewProtocol='TCP',
    NewInternalPort=8000,
    NewInternalClient='192.168.1.3',
    NewEnabled=1,
    NewPortMappingDescription='Test port mapping entry from UPnPpy.',

```

(continues on next page)

(continued from previous page)

```
NewLeaseDuration=0  
)
```

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### U

- [upnpy](#), [12](#)
- [upnpy.exceptions](#), [11](#)
- [upnpy.soap](#), [7](#)
- [upnpy.soap.SOAP](#), [7](#)
- [upnpy.ssdp](#), [10](#)
- [upnpy.ssdp.SSDPDevice](#), [7](#)
- [upnpy.ssdp.SSDPHeader](#), [9](#)
- [upnpy.ssdp.SSDPRequest](#), [9](#)
- [upnpy.upnp](#), [10](#)
- [upnpy.upnp.UPnP](#), [10](#)
- [upnpy.utils](#), [10](#)





## A

ActionNotFoundError, 11  
ArgumentError, 11

## D

discover() (*upnpy.upnp.UPnP.UPnP method*), 10

## G

get\_actions() (*upnpy.ssdp.SSDPDevice.SSDPDevice.Service method*), 8  
get\_friendly\_name() (*upnpy.ssdp.SSDPDevice.SSDPDevice method*), 9  
get\_igd() (*upnpy.upnp.UPnP.UPnP method*), 10  
get\_input\_arguments() (*upnpy.ssdp.SSDPDevice.SSDPDevice.Service.Action method*), 8  
get\_output\_arguments() (*upnpy.ssdp.SSDPDevice.SSDPDevice.Service.Action method*), 8  
get\_services() (*upnpy.ssdp.SSDPDevice.SSDPDevice method*), 9

## I

IGDError, 11

## M

m\_search() (*upnpy.ssdp.SSDPRequest.SSDPRequest method*), 9  
make\_http\_request() (*in module upnpy.utils*), 10

## N

NotAvailableError, 12  
notify() (*upnpy.ssdp.SSDPRequest.SSDPRequest method*), 9  
NotRetrievedError, 12

## P

parse\_device\_type() (*in module upnpy.utils*), 11  
parse\_http\_header() (*in module upnpy.utils*), 11

parse\_service\_id() (*in module upnpy.utils*), 11

## S

send() (*in module upnpy.soap.SOAP*), 7  
ServiceNotFoundError, 12  
set\_header() (*upnpy.ssdp.SSDPHeader.SSDPHeader method*), 9  
set\_headers() (*upnpy.ssdp.SSDPHeader.SSDPHeader method*), 9  
set\_method() (*upnpy.ssdp.SSDPHeader.SSDPHeader method*), 9  
SOAPError, 12  
SSDPDevice (*class in upnpy.ssdp.SSDPDevice*), 7  
SSDPDevice.Service (*class in upnpy.ssdp.SSDPDevice*), 8  
SSDPDevice.Service.Action (*class in upnpy.ssdp.SSDPDevice*), 8  
SSDPDevice.Service.Action.Argument (*class in upnpy.ssdp.SSDPDevice*), 8  
SSDPDevice.Service.StateVariable (*class in upnpy.ssdp.SSDPDevice*), 8  
SSDPHeader (*class in upnpy.ssdp.SSDPHeader*), 9  
SSDPRequest (*class in upnpy.ssdp.SSDPRequest*), 9

## U

UPnP (*class in upnpy.upnp.UPnP*), 10  
upnpy (*module*), 12  
upnpy.exceptions (*module*), 11  
upnpy.soap (*module*), 7  
upnpy.soap.SOAP (*module*), 7  
upnpy.ssdp (*module*), 10  
upnpy.ssdp.SSDPDevice (*module*), 7  
upnpy.ssdp.SSDPHeader (*module*), 9  
upnpy.ssdp.SSDPRequest (*module*), 9  
upnpy.upnp (*module*), 10  
upnpy.upnp.UPnP (*module*), 10  
upnpy.utils (*module*), 10