python-zeroconf

Release 0.39.4

Oct 31, 2022

Contents

GitHub (code repository, issues): https://github.com/jstasiak/python-zeroconf

PyPI (installable, stable distributions): https://pypi.org/project/zeroconf. You can install python-zeroconf using pip:

```
pip install zeroconf
```

python-zeroconf works with CPython 3.6+ and PyPy 3 implementing Python 3.6+.

CHAPTER 1

Contents

1.1 python-zeroconf API reference

Multicast DNS Service Discovery for Python, v0.14-wmcbrine Copyright 2003 Paul Scott-Murphy, 2014 William McBrine

This module provides a framework for the use of DNS Service Discovery using IP multicast.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

class zeroconf.**Zeroconf**(*interfaces: Union[List[Union[str, int, Tuple[Tuple[str, int, int], int]]], zeroconf._utils.net.InterfaceChoice] = <InterfaceChoice.All: 2>, unicast: bool = False, ip_version: Optional[zeroconf._utils.net.IPVersion] = None, apple_p2p: bool = False*)

Bases: zeroconf._logger.QuietLogger

Implementation of Zeroconf Multicast DNS Service Discovery

Supports registration, unregistration, queries and browsing.

add_listener(*listener: zeroconf._updates.RecordUpdateListener, question: Union[zeroconf._dns.DNSQuestion, List[zeroconf._dns.DNSQuestion], None]*) → None

Adds a listener for a given question. The listener will have its update_record method called when information is available to answer the question(s).

This function is threadsafe

add_service_listener(*type_: str*, *listener: zeroconf._services.ServiceListener*) → None
Adds a listener for a particular service type. This object will then have its add_service and remove_service methods called when services of that type become available and unavailable.

async_add_listener(*listener: zeroconf._updates.RecordUpdateListener, question: Union[zeroconf._dns.DNSQuestion, List[zeroconf._dns.DNSQuestion], None]*) → None
Adds a listener for a given question. The listener will have its update_record method called when information is available to answer the question(s).

This function is not threadsafe and must be called in the eventloop.

async_check_service(*info: zeroconf._services.info.ServiceInfo*, *allow_name_change: bool*, *cooperating_responders: bool = False*) → None
Checks the network for a unique service name, modifying the ServiceInfo passed in if it is not unique.

async_notify_all() → None
Schedule an async_notify_all.

async_register_service(*info: zeroconf._services.info.ServiceInfo*, *ttl: Optional[int] = None*, *allow_name_change: bool = False*, *cooperating_responders: bool = False*) → Awaitable[T_co]
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service. The name of the service may be changed if needed to make it unique on the network. Additionally multiple cooperating responders can register the same service on the network for resilience (if you want this behavior set *cooperating_responders* to *True*).

async_remove_listener(*listener: zeroconf._updates.RecordUpdateListener*) → None
Removes a listener.

This function is not threadsafe and must be called in the eventloop.

async_send(*out: zeroconf._protocol.outgoing.DNSOutgoing, addr: Optional[str] = None, port: int = 5353, v6_flow_scope: Union[Tuple[()], Tuple[int, int]] = (), transport: Optional[asyncio.transports.DatagramTransport] = None*) → None
Sends an outgoing packet.

async_unregister_all_services() → None
Unregister all registered services.

Unlike async_register_service and async_unregister_service, this method does not return a future and is always expected to be awaited since its only called at shutdown.

async_unregister_service(*info: zeroconf._services.info.ServiceInfo*) → Awaitable[T_co]
Unregister a service.

async_update_service(*info: zeroconf._services.info.ServiceInfo*) → Awaitable[T_co]
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service.

async_wait(*timeout: float*) → None
Calling task waits for a given number of milliseconds or until notified.

async_wait_for_start() → None
Wait for start up for actions that require a running Zeroconf instance.

Throws NotRunningException if the instance is not running or could not be started.

close() → None
Ends the background threads, and prevent this instance from servicing further queries.

This method is idempotent and irreversible.

generate_service_broadcast(*info: zeroconf._services.info.ServiceInfo, ttl: Optional[int], broadcast_addresses: bool = True*) → zeroconf._protocol.outgoing.DNSOutgoing
Generate a broadcast to announce a service.

generate_service_query(*info: zeroconf._services.info.ServiceInfo*) → zeroconf._protocol.outgoing.DNSOutgoing
Generate a query to lookup a service.

generate_unregister_all_services() → Optional[zeroconf._protocol.outgoing.DNSOutgoing]
Generate a DNSOutgoing goodbye for all services and remove them from the registry.

get_service_info(*type_: str, name: str, timeout: int = 3000, question_type: Optional[zeroconf._dns.DNSQuestionType] = None*) → Optional[zeroconf._services.info.ServiceInfo]
Returns network's service information for a particular name and type, or None if no service matches by the timeout, which defaults to 3 seconds.

handle_assembled_query(*packets: List[zeroconf._protocol.incoming.DNSIncoming], addr: str, port: int, transport: asyncio.transports.DatagramTransport, v6_flow_scope: Union[Tuple[()], Tuple[int, int]] = ()*) → None
Respond to a (re)assembled query.

If the protocol recieved packets with the TC bit set, it will wait a bit for the rest of the packets and only call handle_assembled_query once it has a complete set of packets or the timer expires. If the TC bit is not set, a single packet will be in packets.

handle_response(*msg: zeroconf._protocol.incoming.DNSIncoming*) → None
Deal with incoming response packets. All answers are held in the cache, and listeners are notified.

notify_all() → None
Notifies all waiting threads and notify listeners.

register_service(*info: zeroconf._services.info.ServiceInfo, ttl: Optional[int] = None, allow_name_change: bool = False, cooperating_responders: bool = False*) → None
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service. The name of the service may be changed if needed to make it unique on the network. Additionally multiple cooperating responders can register the same service on the network for resilience (if you want this behavior set *cooperating_responders* to *True*).

While it is not expected during normal operation, this function may raise EventLoopBlocked if the underlying call to *register_service* cannot be completed.

remove_all_service_listeners() → None
Removes a listener from the set that is currently listening.

remove_listener(*listener: zeroconf._updates.RecordUpdateListener*) → None
Removes a listener.

This function is threadsafe

remove_service_listener(*listener: zeroconf._services.ServiceListener*) → None
Removes a listener from the set that is currently listening.

send(*out: zeroconf._protocol.outgoing.DNSOutgoing, addr: Optional[str] = None, port: int = 5353, v6_flow_scope: Union[Tuple[()], Tuple[int, int]] = (), transport: Optional[asyncio.transports.DatagramTransport] = None*) → None
Sends an outgoing packet threadsafe.

start() → None
Start Zeroconf.

started
Check if the instance has started.

unregister_all_services() → None
Unregister all registered services.

While it is not expected during normal operation, this function may raise EventLoopBlocked if the underlying call to *async_unregister_all_services* cannot be completed.

unregister_service(*info: zeroconf._services.info.ServiceInfo*) → None
Unregister a service.

While it is not expected during normal operation, this function may raise EventLoopBlocked if the underlying call to *async_unregister_service* cannot be completed.

update_service(*info: zeroconf._services.info.ServiceInfo*) → None
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service.

While it is not expected during normal operation, this function may raise EventLoopBlocked if the underlying call to *async_update_service* cannot be completed.

class zeroconf.**ServiceInfo**(*type_: str*, *name: str*, *port: Optional[int] = None*, *weight: int = 0*, *priority: int = 0*, *properties: Union[bytes, Dict[KT, VT]] = b''*, *server: Optional[str] = None*, *host_ttl: int = 120*, *other_ttl: int = 4500*, ***, *addresses: Optional[List[bytes]] = None*, *parsed_addresses: Optional[List[str]] = None*, *interface_index: Optional[int] = None*)
Bases: zeroconf._updates.RecordUpdateListener

Service information.

Constructor parameters are as follows:

- *type_*: fully qualified service type name
- *name*: fully qualified service name
- *port*: port that the service runs on
- *weight*: weight of the service
- *priority*: priority of the service
- *properties*: dictionary of properties (or a bytes object holding the contents of the *text* field). converted to str and then encoded to bytes using UTF-8. Keys with *None* values are converted to value-less attributes.
- *server*: fully qualified name for service host (defaults to name)
- *host_ttl*: ttl used for A/SRV records
- *other_ttl*: ttl used for PTR/TXT records
- *addresses* and *parsed_addresses*: List of IP addresses (either as bytes, network byte order, or in parsed form as text; at most one of those parameters can be provided)
- interface_index: scope_id or zone_id for IPv6 link-local addresses i.e. an identifier of the interface where the peer is connected to

addresses
IPv4 addresses of this service.

Only IPv4 addresses are returned for backward compatibility. Use *addresses_by_version()* or *parsed_addresses()* to include IPv6 addresses as well.

addresses_by_version(*version: zeroconf._utils.net.IPVersion*) → List[bytes]
List addresses matching IP version.

async_request(*zc: Zeroconf, timeout: float, question_type: Optional[zeroconf._dns.DNSQuestionType] = None*) → bool
Returns true if the service could be discovered on the network, and updates this object with details discovered.

async_update_records(*zc: Zeroconf, now: float, records: List[zeroconf._updates.RecordUpdate]*) → None
Updates service information from a DNS record.

This method will be run in the event loop.

dns_addresses(*override_ttl: Optional[int] = None, version: zeroconf._utils.net.IPVersion = <IPVersion.All: 3>, created: Optional[float] = None*) → List[zeroconf._dns.DNSAddress]
Return matching DNSAddress from ServiceInfo.

dns_pointer(*override_ttl: Optional[int] = None, created: Optional[float] = None*) → zeroconf._dns.DNSPointer
Return DNSPointer from ServiceInfo.

dns_service(*override_ttl: Optional[int] = None, created: Optional[float] = None*) → zeroconf._dns.DNSService
Return DNSService from ServiceInfo.

dns_text(*override_ttl: Optional[int] = None, created: Optional[float] = None*) → zeroconf._dns.DNSText
Return DNSText from ServiceInfo.

generate_request_query(*zc: Zeroconf, now: float, question_type: Optional[zeroconf._dns.DNSQuestionType] = None*) → zeroconf._protocol.outgoing.DNSOutgoing
Generate the request query.

get_name() → str
Name accessor

load_from_cache(*zc: Zeroconf*) → bool
Populate the service info from the cache.

This method is designed to be threadsafe.

name
The name of the service.

parsed_addresses(*version: zeroconf._utils.net.IPVersion = <IPVersion.All: 3>*) → List[str]
List addresses in their parsed string form.

parsed_scoped_addresses(*version: zeroconf._utils.net.IPVersion = <IPVersion.All: 3>*) → List[str]
Equivalent to parsed_addresses, with the exception that IPv6 Link-Local addresses are qualified with %<interface_index> when available

properties
If properties were set in the constructor this property returns the original dictionary of type *Dict[Union[bytes, str], Any]*.

If properties are coming from the network, after decoding a TXT record, the keys are always bytes and the values are either bytes, if there was a value, even empty, or *None*, if there was none. No further decoding is attempted. The type returned is *Dict[bytes, Optional[bytes]]*.

request(*zc: Zeroconf, timeout: float, question_type: Optional[zeroconf._dns.DNSQuestionType] = None*) → bool
Returns true if the service could be discovered on the network, and updates this object with details discovered.

While it is not expected during normal operation, this function may raise EventLoopBlocked if the underlying call to *async_request* cannot be completed.

update_record(*zc: Zeroconf, now: float, record: Optional[zeroconf._dns.DNSRecord]*) → None
Updates service information from a DNS record.

This method is deprecated and will be removed in a future version. update_records should be implemented instead.

This method will be run in the event loop.

class zeroconf.**ServiceBrowser**(*zc: Zeroconf, type_: Union[str, list], handlers: Union[zeroconf._services.ServiceListener, List[Callable[[...], None]], None] = None, listener: Optional[zeroconf._services.ServiceListener] = None, addr: Optional[str] = None, port: int = 5353, delay: int = 1000, question_type: Optional[zeroconf._dns.DNSQuestionType] = None*)
Bases: zeroconf._services.browser._ServiceBrowserBase, threading.Thread

Used to browse for a service of a specific type.

The listener object will have its add_service() and remove_service() methods called when this browser discovers changes in the services availability.

cancel() → None
Cancel the browser.

run() → None
Run the browser thread.

class zeroconf.**DNSQuestionType**
Bases: enum.Enum

An MDNS question type.

"QU" - questions requesting unicast responses "QM" - questions requesting multicast responses https://datatracker.ietf.org/doc/html/rfc6762#section-5.4

QM = 2

QU = 1

class zeroconf.**InterfaceChoice**
Bases: enum.Enum

An enumeration.

All = 2

Default = 1

class zeroconf.**ServiceStateChange**
Bases: enum.Enum

An enumeration.

Added = 1

Removed = 2

Updated = 3

class zeroconf.**IPVersion**
Bases: enum.Enum

An enumeration.

All = 3

V4Only = 1

V6Only = 2

class zeroconf.**ZeroconfServiceTypes**
Bases: zeroconf._services.ServiceListener

Return all of the advertised services on any local networks

add_service(*zc: zeroconf._core.Zeroconf*, *type_: str*, *name: str*) → None
Service added.

classmethod find(*zc: Optional[zeroconf._core.Zeroconf] = None, timeout: Union[int, float] = 5, interfaces: Union[List[Union[str, int, Tuple[Tuple[str, int, int], int]]], zeroconf._utils.net.InterfaceChoice] = <InterfaceChoice.All: 2>, ip_version: Optional[zeroconf._utils.net.IPVersion] = None*) → Tuple[str, ...]
Return all of the advertised services on any local networks.

Parameters

- **zc** – Zeroconf() instance. Pass in if already have an instance running or if non-default interfaces are needed
- **timeout** – seconds to wait for any responses
- **interfaces** – interfaces to listen on.
- **ip_version** – IP protocol version to use.

Returns tuple of service type strings

remove_service(*zc: zeroconf._core.Zeroconf*, *type_: str*, *name: str*) → None
Service removed.

update_service(*zc: zeroconf._core.Zeroconf*, *type_: str*, *name: str*) → None
Service updated.

class zeroconf.**RecordUpdate**(*new*, *old*)
Bases: tuple

new
Alias for field number 0

old
Alias for field number 1

class zeroconf.**RecordUpdateListener**
Bases: object

Base call for all record listeners.

All listeners passed to async_add_listener should use RecordUpdateListener as a base class. In the future it will be required.

async_update_records(*zc: Zeroconf, now: float, records: List[zeroconf._updates.RecordUpdate]*) → None
Update multiple records in one shot.

All records that are received in a single packet are passed to update_records.

This implementation is a compatiblity shim to ensure older code that uses RecordUpdateListener as a base class will continue to get calls to update_record. This method will raise NotImplementedError in a future version.

At this point the cache will not have the new records

Records are passed as a list of RecordUpdate. This allows consumers of async_update_records to avoid cache lookups.

This method will be run in the event loop.

async_update_records_complete() → None
Called when a record update has completed for all handlers.

At this point the cache will have the new records.

This method will be run in the event loop.

update_record(*zc: Zeroconf*, *now: float*, *record: zeroconf._dns.DNSRecord*) → None
Update a single record.

This method is deprecated and will be removed in a future version. update_records should be implemented instead.

zeroconf.**current_time_millis**() → float
Current time in milliseconds.

The current implemention uses *time.monotonic* but may change in the future.

exception zeroconf.**Error**
Bases: Exception

Base class for all zeroconf exceptions.

exception zeroconf.**AbstractMethodException**
Bases: zeroconf._exceptions.Error

Exception when a required method is not implemented.

exception zeroconf.**BadTypeInNameException**
Bases: zeroconf._exceptions.Error

Exception when the type in a name is invalid.

exception zeroconf.**EventLoopBlocked**
Bases: zeroconf._exceptions.Error

Exception when the event loop is blocked.

This exception is never expected to be thrown during normal operation. It should only happen when the cpu is maxed out or there is something blocking the event loop.

exception zeroconf.**IncomingDecodeError**
Bases: zeroconf._exceptions.Error

Exception when there is invalid data in an incoming packet.

exception zeroconf.**NamePartTooLongException**
Bases: zeroconf._exceptions.Error

Exception when the name is too long.

exception zeroconf.**NonUniqueNameException**
Bases: zeroconf._exceptions.Error

Exception when the name is already registered.

exception zeroconf.**NotRunningException**
Bases: zeroconf._exceptions.Error

Exception when an action is called with a zeroconf instance that is not running.

The instance may not be running because it was already shutdown or startup has failed in some unexpected way.

exception zeroconf.**ServiceNameAlreadyRegistered**
Bases: zeroconf._exceptions.Error

Exception when a service name is already registered.

Multicast DNS Service Discovery for Python, v0.14-wmcbrine Copyright 2003 Paul Scott-Murphy, 2014 William McBrine

This module provides a framework for the use of DNS Service Discovery using IP multicast.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

class zeroconf.asyncio.**AsyncZeroconf**(*interfaces: Union[List[Union[str, int, Tuple[Tuple[str, int, int], int]]], zeroconf._utils.net.InterfaceChoice] = <InterfaceChoice.All: 2>, unicast: bool = False, ip_version: Optional[zeroconf._utils.net.IPVersion] = None, apple_p2p: bool = False, zc: Optional[zeroconf._core.Zeroconf] = None*)
Bases: object

Implementation of Zeroconf Multicast DNS Service Discovery

Supports registration, unregistration, queries and browsing.

The async version is currently a wrapper around Zeroconf which is now also async. It is expected that an asyncio event loop is already running before creating the AsyncZeroconf object.

async_add_service_listener(*type_: str*, *listener: zeroconf._services.ServiceListener*) → None
Adds a listener for a particular service type. This object will then have its add_service and remove_service methods called when services of that type become available and unavailable.

async_close() → None
Ends the background threads, and prevent this instance from servicing further queries.

async_get_service_info(*type_: str*, *name: str*, *timeout: int = 3000*, *question_type: Optional[zeroconf._dns.DNSQuestionType] = None*) → Optional[zeroconf.asyncio.AsyncServiceInfo]
Returns network's service information for a particular name and type, or None if no service matches by the timeout, which defaults to 3 seconds.

async_register_service(*info: zeroconf._services.info.ServiceInfo*, *ttl: Optional[int] = None*, *allow_name_change: bool = False*, *cooperating_responders: bool = False*) → Awaitable[T_co]
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service. The name of the service may be changed if needed to make it unique on the network. Additionally multiple cooperating responders can register the same service on the network for resilience (if you want this behavior set *cooperating_responders* to *True*).

The service will be broadcast in a task. This task is returned and therefore can be awaited if necessary.

async_remove_all_service_listeners() → None
Removes a listener from the set that is currently listening.

async_remove_service_listener(*listener: zeroconf._services.ServiceListener*) → None
Removes a listener from the set that is currently listening.

async_unregister_all_services() → None
Unregister all registered services.

Unlike async_register_service and async_unregister_service, this method does not return a future and is always expected to be awaited since its only called at shutdown.

async_unregister_service(*info: zeroconf._services.info.ServiceInfo*) → Awaitable[T_co]
Unregister a service.

The service will be broadcast in a task. This task is returned and therefore can be awaited if necessary.

async_update_service(*info: zeroconf._services.info.ServiceInfo*) → Awaitable[T_co]
Registers service information to the network with a default TTL. Zeroconf will then respond to requests for information for that service.

The service will be broadcast in a task. This task is returned and therefore can be awaited if necessary.

class zeroconf.asyncio.**AsyncServiceInfo**(*type_: str*, *name: str*, *port: Optional[int] = None*, *weight: int = 0*, *priority: int = 0*, *properties: Union[bytes, Dict[KT, VT]] = b''*, *server: Optional[str] = None*, *host_ttl: int = 120*, *other_ttl: int = 4500*, ***, *addresses: Optional[List[bytes]] = None*, *parsed_addresses: Optional[List[str]] = None*, *interface_index: Optional[int] = None*)
Bases: zeroconf._services.info.ServiceInfo

An async version of ServiceInfo.

class zeroconf.asyncio.**AsyncServiceBrowser**(*zeroconf: zeroconf._core.Zeroconf*, *type_: Union[str, list]*, *handlers: Union[zeroconf._services.ServiceListener, List[Callable[[...], None]], None] = None*, *listener: Optional[zeroconf._services.ServiceListener] = None*, *addr: Optional[str] = None*, *port: int = 5353*, *delay: int = 1000*, *question_type: Optional[zeroconf._dns.DNSQuestionType] = None*)
Bases: zeroconf._services.browser._ServiceBrowserBase

Used to browse for a service for specific type(s).

Constructor parameters are as follows:

- *zc*: A Zeroconf instance
- *type_*: fully qualified service type name
- *handler*: ServiceListener or Callable that knows how to process ServiceStateChange events
- *listener*: ServiceListener
- *addr*: address to send queries (will default to multicast)
- *port*: port to send queries (will default to mdns 5353)

- *delay*: The initial delay between answering questions
- *question_type*: The type of questions to ask (DNSQuestionType.QM or DNSQuestionType.QU)

The listener object will have its add_service() and remove_service() methods called when this browser discovers changes in the services availability.

async_cancel() → None
Cancel the browser.

class zeroconf.asyncio.**AsyncZeroconfServiceTypes**
Bases: zeroconf._services.types.ZeroconfServiceTypes

An async version of ZeroconfServiceTypes.

classmethod async_find(*aiozc: Optional[AsyncZeroconf] = None, timeout: Union[int, float] = 5, interfaces: Union[List[Union[str, int, Tuple[Tuple[str, int, int], int]]], zeroconf._utils.net.InterfaceChoice] = <InterfaceChoice.All: 2>, ip_version: Optional[zeroconf._utils.net.IPVersion] = None*) → Tuple[str, ...]
Return all of the advertised services on any local networks.

Parameters

- **aiozc** – AsyncZeroconf() instance. Pass in if already have an instance running or if non-default interfaces are needed
- **timeout** – seconds to wait for any responses
- **interfaces** – interfaces to listen on.
- **ip_version** – IP protocol version to use.

Returns tuple of service type strings

See the project's README for more information.

Python Module Index

Z

Index