WSDiscovery

Release 2.0

L.A. Fernando, Andrei Kopats, Pieter Jordaan, Petri Savolainen, M

Sep 22, 2023

TABLE OF CONTENTS:

This package provides a short introduction to WS-Discovery, simple command-line tools for discovering and publishing services, and a framework for building WS-Discovery service publishing or discovery applications.

CHAPTER

ONE

WS-DISCOVERY IN A NUTSHELL

WS-Discovery (Web Services Discovery) is a standard widely used by network cameras and other connected devices such as printers. It is based on uni- and multicast SOAP over UDP.

1.1 WS-Discovery message semantics

WS-Discovery defines the following **message semantics** for managing and discovering the availability of networked services.

See *WS-Discovery terms* for explanation of some common terms.

Hello A service *must* send a one-way multicast Hello message when it joins a network, or its metadata changes.

Probe To discover services, optionally limited to a particular service type or scope, a client sends a Probe message. Probe can be unicast or multicast.

Probe match When a service receives a matching Probe, it *must* respond with a Probe Match message.

Resolve A client may send a one-way multicast Resolve message to locate service address(es).

Resolve match When a service matches a Resolve message, it *must* respond with a unicast Resolve Match message.

Bye A service *should* send a one-way multicast Bye message when preparing to leave a network.

1.2 WS-Discovery terms

Action In WS-Addressing, Action (URI) identifies the semantics of the message.

QName A name used in XML is uniquely qualified when it's associated with a namespace (URI) it belongs to. This package provides a *QName representation* implementation.

Scope Scopes are common identifiers used for organizing web services into logical groups, serving a purpose similar to that of categories or tags. This package provides a *Service Scope representation* implementation.

EPR An Endpoint reference is an URI (Uniform Resource Identifier) that identifies a SOAP resource such as a WS-Discovery service. A EPR is included in a *Probe match* message. A *Resolve* message can then be used to retrieve actual network address for service.

Envelope SOAP messages are wrapped in so-called envelopes. This package provides a *WS-Discovery message envelope & factories* implementation.

For details, see the full WS-Discovery specification.

CHAPTER

TWO

COMMAND-LINE UTILITIES

There are two command-line tools available for discovering and publishing services.

2.1 wsdiscover

Discover services using WS-Discovery

```
wsdiscover [OPTIONS]
```

Options

-s, --scope <scope>
Full scope URI, eg. onvif://www.onvif.org/Model/

-a, --address <address>
Service address

-p, --port <port>
Service port

-l, --loglevel <loglevel>
Log level; one of INFO, DEBUG, WARNING, ERROR

-c, --capture <capture>
Capture messages to a file

2.2 wspublish

Publish services using WS-Discovery

```
wspublish [OPTIONS]
```

Options

-s, --scope <scope>
Full scope URI, eg. onvif://www.onvif.org/Model/

-t, --typename <typename>
Qualified type name, eg. https://myservicesns:myservice_type

-a, --address <address>
Service IP address

-p, --port <port>
Service port

-l, --loglevel <loglevel>
Log level; one of INFO, DEBUG, WARNING, ERROR

-c, --capture <capture>
Capture messages to a file

CHAPTER

THREE

FRAMEWORK FOR DISCOVERY APPLICATIONS

Reusable facilities for implementing networked service publishing and discovery applications.

3.1 Service publishing implementation

class wsdiscovery.publishing.**ThreadedWSPublishing**(***kwargs*)

Bases: *wsdiscovery.threaded.ThreadedNetworking*, wsdiscovery.publishing.Publishing, *wsdiscovery.daemon.Daemon*

threaded service publishing

clearLocalServices()

send Bye messages for the services and remove them

publishService(*types*, *scopes*, *xAddrs*)

Publish a service with the given TYPES, SCOPES and XAddrs (service addresses)

if xAddrs contains item, which includes {ip} pattern, one item per IP address will be sent

start()

start networking - should be called before using other methods

stop()

cleans up and stops networking

3.2 Service discovery implementation

class wsdiscovery.discovery.**ThreadedWSDiscovery**(***kwargs*)

Bases: *wsdiscovery.daemon.Daemon*, wsdiscovery.discovery.Discovery, *wsdiscovery.threaded.ThreadedNetworking*

Full threaded service discovery implementation

clearRemoteServices()

clears remotely discovered services

searchServices(*types=None*, *scopes=None*, *address=None*, *port=None*, *timeout=3*)

search for services given the TYPES and SCOPES in a given TIMEOUT

setRemoteServiceByeCallback(*cb*)

Set callback, which will be called when new service appeared online and sent Hi message Service is passed as a parameter to the callback Set None to disable callback

setRemoteServiceHelloCallback(*cb*, *types=None*, *scopes=None*)
Set callback, which will be called when new service appeared online and sent Hi message

typesFilter and scopesFilter might be list of types and scopes. If filter is set, callback is called only for Hello messages, which match filter

Set None to disable callback

start()
start networking - should be called before using other methods

stop()
cleans up and stops networking

3.3 Networking daemon base classes

The framework decouples WS-Discovery messaging functionality from the actual networking system implementation.

A set of base classes are provided for creating networked service discovery and/or publishing implementations:

3.3.1 Generic networking base daemon

Generic networking-agnostic WS-Discovery messaging daemon mixin implementation.

class wsdiscovery.daemon.**Daemon**(*uuid_=None*, *capture=None*, *ttl=1*, ***kwargs*)
generic WS-Discovery messaging daemon implementation

_sendBye(*service*)

_sendHello(*service*)

_sendProbe(*types=None*, *scopes=None*, *address=None*, *port=None*)

_sendProbeMatch(*services*, *relatesTo*, *addr*)

_sendResolve(*epr*)

_sendResolveMatch(*service*, *relatesTo*, *addr*)

envReceived(*env*, *addr*)

3.3.2 Threaded networking base classes

Threaded networking facilities for implementing threaded WS-Discovery daemons.

class wsdiscovery.threaded.**AddressMonitorThread**(*wsd*)
trigger address change callbacks when local service addresses change

_updateAddrs()

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class wsdiscovery.threaded.**NetworkingThread**(*observer*, *capture=None*)

static _createMulticastInSocket()

static _createMulticastOutSocket(*addr*, *ttl*)

static _makeMreq(*addr*)

_recvMessages()

_sendMsg(*msg*)

_sendPendingMessages()
Method sleeps, if nothing to do

addMulticastMessage(*env*, *addr*, *port*, *initialDelay=0*)

addSourceAddr(*addr*)
None means 'system default'

addUnicastMessage(*env*, *addr*, *port*, *initialDelay=0*)

join()
Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

removeSourceAddr(*addr*)

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

start()
Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

class wsdiscovery.threaded.**ThreadedNetworking**(***kwargs*)
handle threaded networking start & stop, address add/remove & message sending

_startThreads()

_stopThreads()

addSourceAddr(*addr*)

removeSourceAddr(*addr*)

sendMulticastMessage(*env*, *initialDelay=0*)
handle multicast message sending

sendUnicastMessage(*env*, *host*, *port*, *initialDelay=0*)
handle unicast message sending

start()
start networking - should be called before using other methods

stop()
cleans up and stops networking

class wsdiscovery.threaded.**_StoppableDaemonThread**
Stoppable daemon thread.

run() method shall exit, when self._quitEvent.wait() returned True

schedule_stop()
Schedule stopping the thread. Use join() to wait, until thread really has been stopped

3.4 WS-Discovery message facilities

3.4.1 WS-Discovery message envelope & factories

SOAP envelope implementation.

class wsdiscovery.envelope.**SoapEnvelope**
envelope implementation

getAction()

getEPR()

getInstanceId()

getMessageId()

getMessageNumber()

getMetadataVersion()

getProbeResolveMatches()

getRelatesTo()

getRelationshipType()

getReplyTo()

getScopes()

getSequenceId()

getTo()

getTypes()

getXAddrs()

setAction(*action*)

setEPR(*epr*)

setInstanceId(*instanceId*)

setMessageId(*messageId*)

setMessageNumber(*messageNumber*)

setMetadataVersion(*metadataVersion*)

setProbeResolveMatches(*probeResolveMatches*)

setRelatesTo(*relatesTo*)

setRelationshipType(*relationshipType*)

setReplyTo(*replyTo*)

setScopes(*scopes*)

setSequenceId(*sequenceId*)

setTo(*to*)

setTypes(*types*)

setXAddrs(*xAddrs*)

3.4.2 WS-Discovery messages (de)serialization

Functions to serialize and deserialize messages between SOAP envelope & string representations

wsdiscovery.message.**createSOAPMessage**(*env*)
serialize SOAP envelopes into XML strings

wsdiscovery.message.**parseSOAPMessage**(*data*, *ipAddr*)
deserialize XML message strings into SOAP envelope objects

Serialization & deserialization functions for each message:

Bye message (de)serialization

Serialize & parse WS-Discovery Bye SOAP messages

wsdiscovery.actions.bye.**constructBye**(*service*)
construct an envelope that represents a Bye message

wsdiscovery.actions.bye.**createByeMessage**(*env*)
serialize a SOAP envelope object into a string

wsdiscovery.actions.bye.**parseByeMessage**(*dom*)
parse a XML message into a SOAP envelope object

Hello message (de)serialization

Serialize & parse WS-Discovery Hello SOAP messages

wsdiscovery.actions.hello.**constructHello**(*service*)
construct an envelope that represents a Hello message

wsdiscovery.actions.hello.**createHelloMessage**(*env*)
serialize a SOAP envelope object into a string

wsdiscovery.actions.hello.**parseHelloMessage**(*dom*)
parse a XML message into a SOAP envelope object

Probe message (de)serialization

Serialize & parse WS-Discovery Probe SOAP messages

`wsdiscovery.actions.probe.`**`constructProbe`**(*types*, *scopes*)
construct an envelope that represents a `Probe` message

`wsdiscovery.actions.probe.`**`createProbeMessage`**(*env*)
serialize a SOAP envelope object into a string

`wsdiscovery.actions.probe.`**`parseProbeMessage`**(*dom*)
parse a XML message into a SOAP envelope object

Probe match message (de)serialization

Serialize & parse WS-Discovery Probe Match SOAP messages

`wsdiscovery.actions.probematch.`**`constructProbeMatch`**(*services*, *relatesTo*)
construct an envelope that represents a `Probe Match` message

`wsdiscovery.actions.probematch.`**`createProbeMatchMessage`**(*env*)
serialize a SOAP envelope object into a string

`wsdiscovery.actions.probematch.`**`parseProbeMatchMessage`**(*dom*)
parse a XML message into a SOAP envelope object

Resolve message (de)serialization

Serialize & parse WS-Discovery Resolve SOAP messages

`wsdiscovery.actions.resolve.`**`constructResolve`**(*epr*)
construct an envelope that represents a `Resolve` message

`wsdiscovery.actions.resolve.`**`createResolveMessage`**(*env*)
serialize a SOAP envelope object into a string

`wsdiscovery.actions.resolve.`**`parseResolveMessage`**(*dom*)
parse a XML message into a SOAP envelope object

Resolve match message (de)serialization

Serialize & parse WS-Discovery Resolve Match SOAP messages

`wsdiscovery.actions.resolvematch.`**`constructResolveMatch`**(*service*, *relatesTo*)
construct an envelope that represents a `Resolve Match` message

`wsdiscovery.actions.resolvematch.`**`createResolveMatchMessage`**(*env*)
serialize a SOAP envelope object into a string

`wsdiscovery.actions.resolvematch.`**`parseResolveMatchMessage`**(*dom*)
parse a XML message into a SOAP envelope object

See also:

WS-Discovery message envelope & factories

3.5 Service representation

Discoverable WS-Discovery service.

class wsdiscovery.service.**Service**(*types*, *scopes*, *xAddrs*, *epr*, *instanceId*)
A web service representation implementation

getEPR()
get endpoint reference

getInstanceId()

getMessageNumber()

getMetadataVersion()

getScopes()
get the service scopes

getTypes()
get service types

getXAddrs()
get service network address

incrementMessageNumber()

setEPR(*epr*)
set endpoint reference

setInstanceId(*instanceId*)

setMessageNumber(*messageNumber*)

setMetadataVersion(*metadataVersion*)

setScopes(*scopes*)
set the service scopes

setTypes(*types*)
set service types

setXAddrs(*xAddrs*)
set service network address

3.6 Service Scope representation

Service scopes are used to constrain service discovery.

class wsdiscovery.scope.**Scope**(*value*, *matchBy=None*)
Service scope implementation.

getMatchBy()

getQuotedValue()

getValue()

3.7 QName representation

Qualified name support; see e.g. https://en.wikipedia.org/wiki/QName

class wsdiscovery.qname.**QName**(*namespace*, *localname*, *namespace_prefix=None*)
Qualified name implementation

getFullname()

getLocalname()

getNamespace()

getNamespacePrefix()

3.8 URI parser for scope matching

Module with URI implementation that supports service scope matching

class wsdiscovery.uri.**URI**(*uri*)
URI implementation with additional functionality useful for service scope matching

getAuthority()

getPath()

getPathExQueryFragment()

getScheme()

3.9 UDP communications handling

UDP (User Datagram Protocol) message implementation that helps with management of unicast/multicast semantics and repeat & delay handling.

WS-Discovery spec dictates that the example algorithm provided in SOAP-over-UDP spec is to be used; see

http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html#_Toc229451838

class wsdiscovery.udp.**UDPMessage**(*env*, *addr*, *port*, *msgType*, *initialDelay=0*)
UDP message management implementation

MULTICAST = 'multicast'

UNICAST = 'unicast'

canSend()

getAddr()

getEnv()

getPort()

isFinished()

msgType()

refresh()

3.10 Namespace definitions

SOAP & WS-Discovery XML namespaces used by the package

wsdiscovery.namespaces.**NS_ACTION_BYE = 'http://schemas.xmlsoap.org/ws/2005/04/discovery/Bye**
Bye message namespace

wsdiscovery.namespaces.**NS_ACTION_HELLO = 'http://schemas.xmlsoap.org/ws/2005/04/discovery/H**
Hello message namespace

wsdiscovery.namespaces.**NS_ACTION_PROBE = 'http://schemas.xmlsoap.org/ws/2005/04/discovery/P**
Probe message namespace

wsdiscovery.namespaces.**NS_ACTION_PROBE_MATCH = 'http://schemas.xmlsoap.org/ws/2005/04/disco**
Probe match message namespace

wsdiscovery.namespaces.**NS_ACTION_RESOLVE = 'http://schemas.xmlsoap.org/ws/2005/04/discovery**
Resolve message namespace

wsdiscovery.namespaces.**NS_ACTION_RESOLVE_MATCH = 'http://schemas.xmlsoap.org/ws/2005/04/dis**
Resolve match message namespace

wsdiscovery.namespaces.**NS_ADDRESSING = 'http://schemas.xmlsoap.org/ws/2004/08/addressing'**
Addressing namespace

wsdiscovery.namespaces.**NS_DISCOVERY = 'http://schemas.xmlsoap.org/ws/2005/04/discovery'**
Discovery namespace

wsdiscovery.namespaces.**NS_SOAPENV = 'http://www.w3.org/2003/05/soap-envelope'**
SOAP envelope namespace

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

G

H

I

J

M

N

P

Q

R

S

T

U

W