Zeep Documentation

Release 4.1.0

Michael van Tellingen

Dec 29, 2021

Contents

1 Quick Introduction **3**

2 Installation **5**

3 Getting started **7**

4 A simple use-case **9**

5 User guide **11**
- 5.1 Using Zeep . . . 11
- 5.2 The Client object . . . 11
- 5.3 Settings . . . 14
- 5.4 Transports . . . 15
- 5.5 SOAP headers . . . 18
- 5.6 Datastructures . . . 19
- 5.7 SOAP Attachments (multipart) . . . 22
- 5.8 WS-Addressing (WSA) . . . 22
- 5.9 WS-Security (WSSE) . . . 23
- 5.10 Plugins . . . 24
- 5.11 Helpers . . . 25
- 5.12 Reporting bugs . . . 26

6 API Documentation **29**
- 6.1 Public API . . . 29
- 6.2 Internals . . . 31

7 Changelog **57**
- 7.1 Changelog . . . 57

Python Module Index **71**

Index **73**

A fast and modern Python SOAP client

Highlights:

- Compatible with Python 3.6, 3.7, 3.8 and PyPy
- Build on top of lxml and requests
- Support for Soap 1.1, Soap 1.2 and HTTP bindings
- Support for WS-Addressing headers
- Support for WSSE (UserNameToken / x.509 signing)
- Support for asyncio via httpx
- Experimental support for XOP messages

A simple example:

```
from zeep import Client

client = Client('http://www.webservicex.net/ConvertSpeed.asmx?WSDL')
result = client.service.ConvertSpeed(
    100, 'kilometersPerhour', 'milesPerhour')

assert result == 62.137
```

CHAPTER 1

Quick Introduction

Zeep inspects the WSDL document and generates the corresponding code to use the services and types in the document. This provides an easy to use programmatic interface to a SOAP server.

The emphasis is on SOAP 1.1 and SOAP 1.2, however Zeep also offers support for HTTP Get and Post bindings.

Parsing the XML documents is done by using the lxml library. This is the most performant and compliant Python XML library currently available. This results in major speed benefits when processing large SOAP responses.

The SOAP specifications are unfortunately really vague and leave a lot of things open for interpretation. Due to this there are a lot of WSDL documents available which are invalid or SOAP servers which contain bugs. Zeep tries to be as compatible as possible but there might be cases where you run into problems. Don't hesitate to submit an issue in this case (but please first read *Reporting bugs*).

CHAPTER 2

Installation

Zeep is a pure-python module. This means that there is no C code which needs to be compiled. However the lxml dependency does contain C code since it uses libxml2 and libxslt. For linux/bsd this means you need to install libxml2-dev and libxslt-dev packages. For Windows this is unfortunately a bit more complicated. The easiest way is to install lxml via wheel files since that contains already compiled code for your platform.

To install wheel files you need a recent pip client. See https://pip.pypa.io/en/stable/installing/ how to install pip on your platform.

If you have installed pip then run:

```
pip install zeep
```

Note that the latest version to support Python 2.7, 3.3, 3.4 and 3.5 is Zeep 3.4, install via *pip install zeep==3.4.0*

This assumes that there are wheel files available for the latest lxml release. If that is not the case (https://pypi.python.org/pypi/lxml/) then first install lxml 4.2.5 since that release should have the wheel files for all platforms:

```
pip install lxml==4.2.5 zeep
```

When you want to use wsse.Signature() you will need to install the python xmlsec module. This can be done by installing the `xmlsec` extras:

```
pip install zeep[xmlsec]
```

For the asyncio support in Python 3.6+ the httpx module is required, this can be installed with the `async` extras:

```
pip install zeep[async]
```

CHAPTER 3

Getting started

The first thing you generally want to do is inspect the wsdl file you need to implement. This can be done with:

```
python -mzeep <wsdl>
```

See `python -mzeep --help` for more information about this command.

Note: Zeep follows semver for versioning, however bugs can always occur. So as always pin the version of zeep you tested with (e.g. `zeep==4.1.0`).

CHAPTER 4

A simple use-case

To give you an idea how zeep works a basic example.

```
import zeep

wsdl = 'http://www.soapclient.com/xml/soapresponder.wsdl'
client = zeep.Client(wsdl=wsdl)
print(client.service.Method1('Zeep', 'is cool'))
```

The WSDL used above only defines one simple function (`Method1`) which is made available by zeep via `client.service.Method1`. It takes two arguments and returns a string. To get an overview of the services available on the endpoint you can run the following command in your terminal.

```
python -mzeep http://www.soapclient.com/xml/soapresponder.wsdl
```

Note: Note that unlike suds, zeep doesn't enable caching of the wsdl documents by default. This means that everytime you initialize the client requests are done to retrieve the wsdl contents.

CHAPTER 5

User guide

5.1 Using Zeep

WSDL documents provide a number of operations (functions) per binding. A binding is collection of operations which are called via a specific protocol.

These protocols are generally Soap 1.1 or Soap 1.2. As mentioned before, Zeep also offers experimental support for the Http Get and Http Post bindings. Most of the time this is an implementation detail, Zeep should offer the same API to the user independent of the underlying protocol.

One of the first things you will do if you start developing an interface to a wsdl web service is to get an overview of all available operations and their call signatures. Zeep offers a command line interface to make this easy.

```
python -mzeep http://www.soapclient.com/xml/soapresponder.wsdl
```

See `python -mzeep --help` for more information.

5.2 The Client object

The *`Client`* is the main interface for interacting with a SOAP server. It provides a `service` attribute which references the default binding of the client (via a `ServiceProxy` object). The default binding can be specified when initiating the client by passing the `service_name` and `port_name`. Otherwise the first service and first port within that service are used as the default.

5.2.1 Caching of WSDL and XSD files

When the client is initialized it will automatically retrieve the WSDL file passed as argument. This WSDL file generally references various other WSDL and XSD files. By default Zeep doesn't cache these files but it is however advised to enable this for performance reasons.

Please see *Caching* how to enable this. To make it easy to use the `zeep.CachingClient()` automatically creates a Transport object with SqliteCache enabled.

5.2.2 Configuring the client

The Client class accepts a settings argument to configuring the client. You can initialise the object using the following code:

```
from zeep import Client, Settings

settings = Settings(strict=False, xml_huge_tree=True)
client = Client('http://my-wsdl/wsdl', settings=settings)
```

The settings object is always accessible via the client using `client.settings`. For example:

```
with client.settings(raw_response=True):
    response = client.service.myoperation()
```

Please see *Settings* for more information.

The AsyncClient

The *AsyncClient* allows you to execute operations in an asynchronous fashion. There is one big caveat however: the wsdl documents are still loaded using synchronous methods. The reason for this is that the codebase was originally not written for asynchronous usage and support that is quite a lot of work.

To use async operations you need to use the *AsyncClient()* and the corresponding *AsyncTransport()* (this is the default transport for the *AsyncClient*)

```
client = zeep.AsyncClient("http://localhost:8000/?wsdl")

response = await client.service.myoperation()
```

New in version 4.0.0.

Strict mode

By default zeep will operate in 'strict' mode. This can be disabled if you are working with a SOAP server which is not standards compliant by using the strict setting. See *Settings*. Disabling strict mode will change the following behaviour:

- The XML is parsed with the recover mode enabled
- Nonoptional elements are allowed to be missing in xsd:sequences

Note that disabling strict mode should be considered a last resort since it might result in data-loss between the XML and the returned response.

5.2.3 The ServiceProxy object

The ServiceProxy object is a simple object which will check if an operation exists for attribute or item requested. If the operation exists then it will return an OperationProxy object (callable) which is responsible for calling the operation on the binding.

```
from zeep import Client
from zeep import xsd

client = Client('http://my-endpoint.com/production.svc?wsdl')
```

(continues on next page)

(continued from previous page)

```
# service is a ServiceProxy object.  It will check if there
# is an operation with the name `X` defined in the binding
# and if that is the case it will return an OperationProxy
client.service.X()

# The operation can also be called via an __getitem__ call.
# This is useful if the operation name is not a valid
# python attribute name.
client.service['X-Y']()
```

5.2.4 Using non-default bindings

As mentioned by default Zeep picks the first binding in the WSDL as the default. This binding is available via `client.service`. To use a specific binding you can use the `bind()` method on the client object:

```
from zeep import Client
from zeep import xsd

client = Client('http://my-endpoint.com/production.svc?wsdl')

service2 = client.bind('SecondService', 'Port12')
service2.someOperation(myArg=1)
```

for example, if your wsdl contains these definitions

```
<wsdl:service name="ServiceName">
<wsdl:port name="PortName" binding="tns:BasicHttpsBinding_IServiziPartner">
<soap:address location="https://aaa.bbb.ccc/ddd/eee.svc"/>
</wsdl:port>
<wsdl:port name="PortNameAdmin" binding="tns:BasicHttpsBinding_IServiziPartnerAdmin">
<soap:address location="https://aaa.bbb.ccc/ddd/eee.svc/admin"/>
</wsdl:port>
</wsdl:service>
```

and you need to calls methods defined in **https://aaa.bbb.ccc/ddd/eee.svc/admin** you can do:

```
client = Client("https://www.my.wsdl") # this will use default binding
client_admin = client.bind('ServiceName', 'PortNameAdmin')
client_admin.method1() #this will call method1 defined in service name ServiceName␣
↪and port PortNameAdmin
```

5.2.5 Creating new ServiceProxy objects

There are situations where you either need to change the SOAP address from the one which is defined within the WSDL or the WSDL doesn't define any service elements. This can be done by creating a new ServiceProxy using the `Client.create_service()` method.

```
from zeep import Client
from zeep import xsd

client = Client('http://my-endpoint.com/production.svc?wsdl')
service = client.create_service(
```

(continues on next page)

(continued from previous page)

```
    '{http://my-target-namespace-here}myBinding',
    'http://my-endpoint.com/acceptance/')

service.submit('something')
```

5.2.6 Creating the raw XML documents

When you want zeep to build and return the XML instead of sending it to the server you can use the `Client.create_message()` call. It requires the ServiceProxy as the first argument and the operation name as the second argument.

```
from zeep import Client

client = Client('http://my-endpoint.com/production.svc?wsdl')
node = client.create_message(client.service, 'myOperation', user='hi')
```

5.3 Settings

New in version 3.0.

5.3.1 Context manager

You can set various options directly as attribute on the client or via a context manager.

For example to let zeep return the raw response directly instead of processing it you can do the following:

```
from zeep import Client
from zeep import xsd

client = Client('http://my-endpoint.com/production.svc?wsdl')

with client.settings(raw_response=True):
    response = client.service.myoperation()

    # response is now a regular requests.Response object
    assert response.status_code == 200
    assert response.content
```

5.3.2 API

class zeep.settings.**Settings**(*strict=True*, *raw_response=False*, *force_https=True*, *extra_http_headers=None*, *xml_huge_tree=False*, *forbid_dtd=False*, *forbid_entities=True*, *forbid_external=True*, *xsd_ignore_sequence_order=False*, *tls=NOTHING*)

Parameters

- **strict** (*boolean*) – boolean to indicate if the lxml should be parsed a 'strict'. If false then the recover mode is enabled which tries to parse invalid XML as best as it can.

- **raw_response** – boolean to skip the parsing of the XML response by zeep but instead returning the raw data
- **forbid_dtd** (*bool*) – disallow XML with a <!DOCTYPE> processing instruction
- **forbid_entities** (*bool*) – disallow XML with <!ENTITY> declarations inside the DTD
- **forbid_external** (*bool*) – disallow any access to remote or local resources in external entities or DTD and raising an ExternalReferenceForbidden exception when a DTD or entity references an external resource.
- **xml_huge_tree** – disable lxml/libxml2 security restrictions and support very deep trees and very long text content
- **force_https** (*bool*) – Force all connections to HTTPS if the WSDL is also loaded from an HTTPS endpoint. (default: true)
- **extra_http_headers** – Additional HTTP headers to be sent to the transport. This can be used in combination with the context manager approach to add http headers for specific calls.
- **xsd_ignore_sequence_order** (*boolean*) – boolean to indicate whether to enforce sequence order when parsing complex types. This is a workaround for servers that don't respect sequence order.

5.4 Transports

If you need to change options like cache, timeout or TLS (or SSL) verification you will need to create an instance of the Transport class yourself.

Note: Secure Sockets Layer (SSL) has been deprecated in favor of Transport Layer Security (**TLS**). SSL 2.0 was prohibited in 2011 and SSL 3.0 in June 2015.

5.4.1 TLS verification

If you need to verify the TLS connection (in case you have a self-signed certificate for your host), the best way is to create a `requests.Session` instance and add the information to that Session, so it keeps persistent:

```
from requests import Session
from zeep import Client
from zeep.transports import Transport

session = Session()
session.verify = 'path/to/my/certificate.pem'
transport = Transport(session=session)
client = Client(
    'http://my.own.sslhost.local/service?WSDL',
    transport=transport)
```

Hint: Make sure that the certificate you refer to is a CA_BUNDLE, meaning it contains a root CA and an intermediate CA. Accepted are only X.509 ASCII files (file extension `.pem`, sometimes `.crt`). If you have two different files, you must combine them manually into one.

Alternatively, instead of using `session.verify` you can use `session.cert` if you just want to use an TLS client certificate.

To **disable TLS verification** (not recommended!) you will need to set `verify` to `False`.

```
session = Session()
session.verify = False
```

Or even simpler way:

```
client.transport.session.verify = False
```

Remember: this should be only done for testing purposes. Python's `urllib3` will warn you with a `InsecureRequestWarning`.

See `requests.Session` for further details.

5.4.2 Session timeout

To set a transport timeout for loading wsdl sfn xsd documents, use the *timeout* option. The default timeout is 300 seconds:

```
from zeep import Client
from zeep.transports import Transport

transport = Transport(timeout=10)
client = Client(
    'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
    transport=transport)
```

To pass a timeout to the underlying POST/GET requests, use *operation_timeout*. This defaults to None.

5.4.3 Using HTTP or SOCKS Proxy

By default, zeep uses `requests` as transport layer, which allows to define proxies using the `proxies` attribute of `requests.Session`:

```
from zeep import Client

client = Client(
    'http://my.own.sslhost.local/service?WSDL')

client.transport.session.proxies = {
    # Utilize for all http/https connections
    'http': 'foo.bar:3128',
    'https': 'foo.bar:3128',
    # Utilize for certain URL
    'http://specific.host.example': 'foo.bar:8080',
    # Or use socks5 proxy (requires requests[socks])
    'https://socks5-required.example': 'socks5://foo.bar:8888',
}
```

In order to use **SOCKS** proxies, requests needs to be installed with additional packages (for example `pip install -U requests[socks]`).

5.4.4 Caching

By default zeep doesn't use a caching backend. For performance benefits it is advised to use the SqliteCache backend. It caches the WSDL and XSD files for 1 hour by default. To use the cache backend init the client with:

```
from zeep import Client
from zeep.cache import SqliteCache
from zeep.transports import Transport

transport = Transport(cache=SqliteCache())
client = Client(
    'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
    transport=transport)
```

Changing the SqliteCache settings can be done via:

```
from zeep import Client
from zeep.cache import SqliteCache
from zeep.transports import Transport
cache = SqliteCache(path='/tmp/sqlite.db', timeout=60)
transport = Transport(cache=cache)
client = Client(
    'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
    transport=transport)
```

Another option is to use the InMemoryCache backend. It internally uses a global dict to store urls with the corresponding content.

5.4.5 HTTP Authentication

While some providers incorporate security features in the header of a SOAP message, others use the HTTP Authentication header. In the latter case, you can just create a `requests.Session` object with the auth set and pass it to the Transport class.

```
from requests import Session
from requests.auth import HTTPBasicAuth  # or HTTPDigestAuth, or OAuth1, etc.
from zeep import Client
from zeep.transports import Transport

session = Session()
session.auth = HTTPBasicAuth(user, password)
client = Client('http://my-endpoint.com/production.svc?wsdl',
    transport=Transport(session=session))
```

5.4.6 Async HTTP Authentication

The Async client for zeep uses a different backend, so the setup is different in this case. You will need to use *httpx* to create an `httpx.AsyncClient` object, and pass it to your `zeep.AsyncTransport`.

```
import httpx
import zeep
from zeep.transports import AsyncTransport

USER = 'username'
```

(continues on next page)

(continued from previous page)

```
PASSWORD = 'password'

httpx_client = httpx.AsyncClient(auth=(USER, PASSWORD))

aclient = zeep.AsyncClient(
    "http://my-endpoint.com/production.svc?wsdl",
    transport=AsyncTransport(client=httpx_client)
)
```

5.4.7 Debugging

To see the SOAP XML messages which are sent to the remote server and the response received you can set the Python logger level to DEBUG for the `zeep.transports` module. Since 0.15 this can also be achieved via the *HistoryPlugin*.

```
import logging.config

logging.config.dictConfig({
    'version': 1,
    'formatters': {
        'verbose': {
            'format': '%(name)s: %(message)s'
        }
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',
        },
    },
    'loggers': {
        'zeep.transports': {
            'level': 'DEBUG',
            'propagate': True,
            'handlers': ['console'],
        },
    }
})
```

5.5 SOAP headers

SOAP headers are generally used for things like authentication. The header elements can be passed to all operations using the `_soapheaders` kwarg.

There are multiple ways to pass a value to the soapheader.

1. When the SOAP header expects a complex type you can either pass a dict or an object created via the `client.get_element()` method.
2. When the header expects a simple type value you can pass it directly to the `_soapheaders` kwarg. (e.g.: `client.service.Method(_soapheaders=1234)`)
3. Creating custom xsd element objects. For example:

```
from zeep import xsd

header = xsd.Element(
    '{http://test.python-zeep.org}auth',
    xsd.ComplexType([
        xsd.Element(
            '{http://test.python-zeep.org}username',
            xsd.String()),
    ])
)
header_value = header(username='mvantellingen')
client.service.Method(_soapheaders=[header_value])
```

4. Another option is to pass an lxml Element object. This is generally useful if the wsdl doesn't define a soap header but the server does expect it.

5.6 Datastructures

5.6.1 Creating objects

Most of the times you need to pass nested data to the soap client. These Complex types can be retrieved using the `client.get_type()` method.

```
from zeep import Client

client = Client('http://my-enterprise-endpoint.com')
order_type = client.get_type('ns0:Order')
order = order_type(number='1234', price=99)
client.service.submit_order(user_id=1, order=order)
```

However instead of creating an object from a type defined in the XSD you can also pass in a dictionary. Zeep will automatically convert this dict to the required object (and nested child objects) during the call.

```
from zeep import Client

client = Client('http://my-enterprise-endpoint.com')
client.service.submit_order(user_id=1, order={
    'number': '1234',
    'price': 99,
})
```

5.6.2 Using factories

When you need to create multiple types the `Client.get_type()` calls to retrieve the type class and then instantiating them can be a bit verbose. To simplify this you can use a factory object.

```
from zeep import Client

client = Client('http://my-enterprise-endpoint.com')
factory = client.type_factory('ns0')

user = factory.User(id=1, name='John')
```

(continues on next page)

(continued from previous page)

```
order = factory.Order(number='1234', price=99)
client.service.submit_order(user=user, order=order)
```

New in version 0.22.

5.6.3 xsd:choice

Mapping the semantics of xsd:choice elements to code is unfortunately pretty difficult. Zeep tries to solve this using two methods:

1. Accepting the elements in the xsd:choice element as kwargs. This only works for simple xsd:choice definitions.
2. Using the special kwarg `_value_N` where the N is the number of the choice in the parent type. This method allows you to pass a list of dicts (when maxOccurs != 1) or a dict directly.

The following examples should illustrate this better.

Simple method

```
<?xml version="1.0"?>
<schema xmlns:tns="http://tests.python-zeep.org/"
        targetNamespace="http://tests.python-zeep.org/">
  <element name='ElementName'>
    <complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <choice>
        <element name="item_1" type="string"/>
        <element name="item_2" type="string"/>
      </choice>
    </complexType>
  </element>
</schema>
```

```
element = client.get_element('ns0:ElementName')
obj = element(item_1='foo')
```

Nested using _value_1

```
<?xml version="1.0"?>
<schema xmlns:tns="http://tests.python-zeep.org/"
        targetNamespace="http://tests.python-zeep.org/">
  <element name='ElementName'>
    <complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <choice maxOccurs="1">
        <sequence>
            <element name="item_1_a" type="string"/>
            <element name="item_1_b" type="string"/>
        </sequence>
      </choice>
    </complexType>
  </element>
</schema>
```

```
element = client.get_element('ns0:ElementName')
obj = element(_value_1={'item_1_a': 'foo', 'item_1_b': 'bar'})
```

Nested list using _value_1

```
<?xml version="1.0"?>
<schema xmlns:tns="http://tests.python-zeep.org/"
        targetNamespace="http://tests.python-zeep.org/">
  <element name='ElementName'>
    <complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <choice maxOccurs="unbounded">
        <element name="item_1" type="string"/>
        <element name="item_2" type="string"/>
      </choice>
    </complexType>
  </element>
</schema>
```

```
element = client.get_element('ns0:ElementName')
obj = element(_value_1=[{'item_1': 'foo'}, {'item_2': 'bar'}])
```

5.6.4 Any objects

Zeep offers full support for xsd:any elements. xsd:any elements are used as a kind of wildcard and basically allows any element to be used. Zeep needs to know the element name you want to serialize, so the value needs to be wrapped in a special object. This is the `AnyObject`. It takes two parameters, the xsd Element first and the value as the second arg.

```
from zeep import Client
from zeep import xsd

client = Client('http://my-entrprisy-endpoint.com')
order_type = client.get_element('ns0:Order')
order = xsd.AnyObject(
  order_type, order_type(number='1234', price=99))
client.service.submit_something(user_id=1, _value_1=order)
```

5.6.5 AnyType objects

xsd:anyType is used as a wildcard type. Where the xsd:Any element allows any element the xsd:anyType allows any type for a specific element. The usage from zeep is almost the same. Instead of passing an `Element` class to the AnyObject an xsd type is passed.

```
from zeep import Client
from zeep import xsd

client = Client('http://my-entrprisy-endpoint.com')
value = xsd.AnyObject(xsd.String(), 'foobar')
client.service.submit_something(user_id=1, my_string=value)
```

5.6.6 SkipValue

Zeep will automatically validate that all the required values are set when calling an operation. If you want to force a value to be ignored and left out of the generated XML then you can assign the `zeep.xsd.SkipValue` constant.

```
from zeep import Client
from zeep import xsd

client = Client('http://my-entrprisy-endpoint.com')
client.service.submit_something(user_id=1, my_string=xsd.SkipValue)
```

5.7 SOAP Attachments (multipart)

If the server responds with a Content-type: multipart, a MessagePack object will be returned. It contains a root object and some attachments.

Example based on https://www.w3.org/TR/SOAP-attachments

```
from zeep import Client

client = Client('http://www.risky-stuff.com/claim.svc?wsdl')

pack = client.service.GetClaimDetails('061400a')

ClaimDetails = pack.root
SignedFormTiffImage = pack.attachments[0].content
CrashPhotoJpeg = pack.attachments[1].content

# Or lookup by content_id
pack.get_by_content_id('<claim061400a.tiff@claiming-it.com>').content
```

5.8 WS-Addressing (WSA)

New in version 0.15.

Zeep offers (experimental) support for the ws-addressing specification. The specification defines some soap:Header elements which basically allows advanced routing of the SOAP messages.

If the WSDL document defines that WSA is required then Zeep will automatically add the required headers to the SOAP envelope. In case you want to customize this then you can add the `WsAddressPlugin` to the `Client.plugins` list.

For example:

```
from zeep import Client
from zeep.wsa import WsAddressingPlugin

client = Client(
    'http://examples.python-zeep.org/basic.wsdl',
    plugins=[WsAddressingPlugin()])
client.service.DoSomething()
```

Note: The support for ws-addressing is experimental. If you encounter any issues then please don't hesitate to create an issue on the github repository.

5.9 WS-Security (WSSE)

WS-Security incorporates security features in the header of a SOAP message.

5.9.1 UsernameToken

The UsernameToken supports both the passwordText and passwordDigest methods:

```
>>> from zeep import Client
>>> from zeep.wsse.username import UsernameToken
>>> client = Client(
...     'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
...     wsse=UsernameToken('username', 'password'))
```

To use the passwordDigest method you need to supply *use_digest=True* to the *UsernameToken* class.

5.9.2 Signature (x509)

To use the wsse.Signature() plugin you will need to install the xmlsec module. See the README for xmlsec for the required dependencies on your platform.

To append the security token as *BinarySecurityToken*, you can use wsse.BinarySignature() plugin.

Example usage A:

```
>>> from zeep import Client
>>> from zeep.wsse.signature import Signature
>>> client = Client(
...     'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
...     wsse=Signature(
...         private_key_filename, public_key_filename,
...         optional_password))
```

Example usage B:

```
>>> from zeep import Client
>>> from zeep.wsse.signature import Signature
>>> from zeep.transports import Transport
>>> from requests import Session
>>> session = Session()
>>> session.cert = '/path/to/ssl.pem'
>>> transport = Transport(session=session)
>>> client = Client(
...     'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
...     transport=transport)
```

5.9.3 UsernameToken and Signature together

To use UsernameToken and Signature together, then you can pass both together to the client in a list

```
>>> from zeep import Client
>>> from zeep.wsse.username import UsernameToken
>>> from zeep.wsse.signature import Signature
>>> user_name_token = UsernameToken('username', 'password')
>>> signature = Signature(private_key_filename, public_key_filename,
...     optional_password)
>>> client = Client(
...     'http://www.webservicex.net/ConvertSpeed.asmx?WSDL',
...     wsse=[user_name_token, signature])
```

5.9.4 UsernameToken with Timestamp token

To use UsernameToken with Timestamp token, first you need an instance of *WSU.Timestamp()*, then extend it with a list containing *WSU.Created()* and *WSU.Expired()* elements, finally pass it as *timestamp_token* keyword argument to *UsernameToken()*.

```
>>> import datetime
>>> from zeep import Client
>>> from zeep.wsse.username import UsernameToken
>>> from zeep.wsse.utils import WSU
>>> timestamp_token = WSU.Timestamp()
>>> today_datetime = datetime.datetime.today()
>>> expires_datetime = today_datetime + datetime.timedelta(minutes=10)
>>> timestamp_elements = [
...         WSU.Created(today_datetime.strftime("%Y-%m-%dT%H:%M:%SZ")),
...         WSU.Expires(expires_datetime.strftime("%Y-%m-%dT%H:%M:%SZ"))
...]
>>> timestamp_token.extend(timestamp_elements)
>>> user_name_token = UsernameToken('username', 'password', timestamp_token=timestamp_
↪token)
>>> client = Client(
...     'http://www.webservicex.net/ConvertSpeed.asmx?WSDL', wsse=user_name_token
...)
```

5.10 Plugins

New in version 0.15.

You can write plugins for zeep which can be used to process/modify data before it is sent to the server (egress) and after it is received (ingress).

Writing a plugin is really simple and best explained via an example.

```
from lxml import etree
from zeep import Plugin

class MyLoggingPlugin(Plugin):

    def ingress(self, envelope, http_headers, operation):
        print(etree.tostring(envelope, pretty_print=True))
```

(continues on next page)

(continued from previous page)

```
        return envelope, http_headers

    def egress(self, envelope, http_headers, operation, binding_options):
        print(etree.tostring(envelope, pretty_print=True))
        return envelope, http_headers
```

The plugin can implement two methods: `ingress` and `egress`. Both methods should always return an envelop (lxml element) and the http headers. The envelope in the `egress` plugin will only contain the body of the soap message. This is important to remember if you want to inspect or do something with the headers.

To register this plugin you need to pass it to the client. Plugins are always executed sequentially.

```
from zeep import Client

client = Client(..., plugins=[MyLoggingPlugin()])
```

5.10.1 HistoryPlugin

New in version 0.15.

The history plugin keep a list of sent and received requests. By default at most one transaction (sent/received) is kept. But this can be changed when you create the plugin by passing the `maxlen` kwarg.

```
from zeep import Client
from zeep.plugins import HistoryPlugin

history = HistoryPlugin()
client = Client(
    'http://examples.python-zeep.org/basic.wsdl',
    plugins=[history])
client.service.DoSomething()

print(history.last_sent)
print(history.last_received)
```

5.11 Helpers

In the *zeep.helpers* module the following helper functions are available:

`zeep.helpers.`**`Nil`**`()`
Return an xsi:nil element

`zeep.helpers.`**`create_xml_soap_map`**`(`*values*`)`
Create an http://xml.apache.org/xml-soap#Map value.

`zeep.helpers.`**`guess_xsd_type`**`(`*obj*`)`
Return the XSD Type for the given object

`zeep.helpers.`**`serialize_object`**`(`*obj*, *target_cls=<class 'collections.OrderedDict'>*`)`
Serialize zeep objects to native python data structures

5.12 Reporting bugs

The SOAP specifications are pretty bad and unclear for a lot of use-cases. This results in a lot of (older) SOAP servers which don't implement the specifications correctly (or implement them in a way Zeep doesn't expect). Of course there is also a good chance that Zeep doesn't implement something correctly ;-) I'm always interested in the latter.

Since Zeep is a module I've created and currently maintain mostly in my spare time I need as much information as possible to quickly analyze/fix issues.

There are basically three majors parts where bugs can happen, these are:

1. Parsing the WSDL
2. Creating the XML for the request
3. Parsing the XML from the response

The first one is usually pretty easy to debug if the WSDL is publicly accessible. If that isn't the case then you might be able to make it anonymous.

5.12.1 Required information

Please provide the following information:

1. The version of zeep (or if you are running master the commit hash/date)
2. The WSDL you are using
3. An example script (please see below)

5.12.2 Errors when creating the request

Create a new python script using the code below. The first argument to the `create_message()` method is the name of the operation/method and after that the args / kwargs you normally pass.

```
from lxml import etree
from zeep import Client

client = Client('YOUR-WSDL')

# client.service.OPERATION_NAME(*args, **kwargs) becomes
node = client.create_message(
    client.service, 'OPERATION_NAME', *args, **kwargs)

print(etree.tostring(node, pretty_print=True))
```

5.12.3 Errors when parsing the response

The first step is to retrieve the XML which is returned from the server, You need to enable debugging for this. Please see *Debugging* for a detailed description.

The next step is to create a python script which exposes the problem, an example is the following.

```
import pretend  # pip install pretend

from zeep import Client
from zeep.transports import Transport

# Replace YOUR-WSDL and OPERATION_NAME with the wsdl url
# and the method name you are calling. The response
# needs to be set in the content="""  """ var.

client = Client('YOUR-WSDL')
response = pretend.stub(
    status_code=200,
    headers={},
    content="""
        <!-- The response from the server -->
    """)

operation = client.service._binding._operations['OPERATION_NAME']
result = client.service._binding.process_reply(
    client, operation, response)

print(result)
```

CHAPTER 6

API Documentation

6.1 Public API

6.1.1 Client

class zeep.**Client**(*wsdl*, *wsse=None*, *transport=None*, *service_name=None*, *port_name=None*, *plugins=None*, *settings=None*)

The zeep Client.

Parameters

- **wsdl** –
- **wsse** –
- **transport** – Custom transport class.
- **service_name** – The service name for the service binding. Defaults to the first service in the WSDL document.
- **port_name** – The port name for the default binding. Defaults to the first port defined in the service element in the WSDL document.
- **plugins** – a list of Plugin instances
- **settings** – a zeep.Settings() object

bind(*service_name: Optional[str] = None*, *port_name: Optional[str] = None*)

Create a new ServiceProxy for the given service_name and port_name.

The default ServiceProxy instance (*self.service*) always referes to the first service/port in the wsdl Document. Use this when a specific port is required.

create_message(*service*, *operation_name*, **args*, ***kwargs*)

Create the payload for the given operation.

Return type lxml.etree._Element

create_service(*binding_name*, *address*)
Create a new ServiceProxy for the given binding name and address.

Parameters

- **binding_name** – The QName of the binding
- **address** – The address of the endpoint

get_element(*name*)
Return the element for the given qualified name.

Return type zeep.xsd.Element

get_type(*name*)
Return the type for the given qualified name.

Return type zeep.xsd.ComplexType or zeep.xsd.AnySimpleType

service
The default ServiceProxy instance

Return type ServiceProxy

set_default_soapheaders(*headers*)
Set the default soap headers which will be automatically used on all calls.

Note that if you pass custom soapheaders using a list then you will also need to use that during the operations. Since mixing these use cases isn't supported (yet).

set_ns_prefix(*prefix*, *namespace*)
Set a shortcut for the given namespace.

type_factory(*namespace*)
Return a type factory for the given namespace.

Example:

```
factory = client.type_factory('ns0')
user = factory.User(name='John')
```

Return type Factory

6.1.2 Transport

class zeep.**Transport**(*cache=None*, *timeout=300*, *operation_timeout=None*, *session=None*)
The transport object handles all communication to the SOAP server.

Parameters

- **cache** – The cache object to be used to cache GET requests
- **timeout** – The timeout for loading wsdl and xsd documents.
- **operation_timeout** – The timeout for operations (POST/GET). By default this is None (no timeout).
- **session** – A `request.Session()` object (optional)

get(*address*, *params*, *headers*)
Proxy to requests.get()

Parameters

- **address** – The URL for the request
- **params** – The query parameters
- **headers** – a dictionary with the HTTP headers.

load(*url*)
Load the content from the given URL

post(*address*, *message*, *headers*)
Proxy to requests.posts()

Parameters

- **address** – The URL for the request
- **message** – The content for the body
- **headers** – a dictionary with the HTTP headers.

post_xml(*address*, *envelope*, *headers*)
Post the envelope xml element to the given address with the headers.

This method is intended to be overriden if you want to customize the serialization of the xml element. By default the body is formatted and encoded as utf-8. See `zeep.wsdl.utils.etree_to_string`.

settings(*timeout=None*)
Context manager to temporarily overrule options.

Example:

```
transport = zeep.Transport()
with transport.settings(timeout=10):
    client.service.fast_call()
```

Parameters **timeout** – Set the timeout for POST/GET operations (not used for loading external WSDL or XSD documents)

6.1.3 AnyObject

class zeep.**AnyObject**(*xsd_object*, *value*)
Create an any object

Parameters

- **xsd_object** – the xsd type
- **value** – The value

6.2 Internals

6.2.1 zeep.wsdl

The wsdl module is responsible for parsing the WSDL document. This includes the bindings and messages.

The structure and naming of the modules and classses closely follows the WSDL 1.1 specification.

The serialization and deserialization of the SOAP/HTTP messages is done by the zeep.wsdl.messages modules.

6.2.2 zeep.wsdl.wsdl

class zeep.wsdl.wsdl.**Definition**(*wsdl*, *doc*, *location*)
The Definition represents one wsdl:definition within a Document.

Parameters **wsdl** – The wsdl

parse_binding(*doc: lxml.etree._Element*) → Dict[str, Type[zeep.wsdl.definitions.Binding]]
Parse the binding elements and return a dict of bindings.

Currently supported bindings are Soap 1.1, Soap 1.2., HTTP Get and HTTP Post. The detection of the type of bindings is done by the bindings themselves using the introspection of the xml nodes.

Definition:

```
<wsdl:definitions .... >
    <wsdl:binding name="nmtoken" type="qname"> *
        <-- extensibility element (1) --> *
        <wsdl:operation name="nmtoken"> *
           <-- extensibility element (2) --> *
           <wsdl:input name="nmtoken"? > ?
               <-- extensibility element (3) -->
           </wsdl:input>
           <wsdl:output name="nmtoken"? > ?
               <-- extensibility element (4) --> *
           </wsdl:output>
           <wsdl:fault name="nmtoken"> *
               <-- extensibility element (5) --> *
           </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

Parameters **doc** (*lxml.etree._Element*) – The source document

Returns Dictionary with binding name as key and Binding instance as value

Return type dict

parse_imports(*doc*)
Import other WSDL definitions in this document.

Note that imports are non-transitive, so only import definitions which are defined in the imported document and ignore definitions imported in that document.

This should handle recursive imports though:

A -> B -> A A -> B -> C -> A

Parameters **doc** (*lxml.etree._Element*) – The source document

parse_messages(*doc: lxml.etree._Element*)
Definition:

```
<definitions .... >
    <message name="nmtoken"> *
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </message>
</definitions>
```

Parameters **doc** (*lxml.etree._Element*) – The source document

parse_ports(*doc: lxml.etree._Element*) → Dict[str, zeep.wsdl.definitions.PortType]
Return dict with *PortType* instances as values

Definition:

```
<wsdl:definitions .... >
    <wsdl:portType name="nmtoken">
        <wsdl:operation name="nmtoken" .... /> *
    </wsdl:portType>
</wsdl:definitions>
```

Parameters **doc** (*lxml.etree._Element*) – The source document

parse_service(*doc: lxml.etree._Element*) → Dict[str, zeep.wsdl.definitions.Service]
Definition:

```
<wsdl:definitions .... >
    <wsdl:service .... > *
        <wsdl:port name="nmtoken" binding="qname"> *
           <-- extensibility element (1) -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Parameters **doc** (*lxml.etree._Element*) – The source document

parse_types(*doc*)
Return an xsd.Schema() instance for the given wsdl:types element.

If the wsdl:types contain multiple schema definitions then a new wrapping xsd.Schema is defined with xsd:import statements linking them together.

If the wsdl:types doesn't container an xml schema then an empty schema is returned instead.

Definition:

```
<definitions .... >
    <types>
        <xsd:schema .... />*
    </types>
</definitions>
```

Parameters **doc** (*lxml.etree._Element*) – The source document

resolve_imports() → None
Resolve all root elements (types, messages, etc).

class zeep.wsdl.wsdl.**Document**(*location, transport: Type[Transport], base=None, settings=None*)
A WSDL Document exists out of one or more definitions.

There is always one 'root' definition which should be passed as the location to the Document. This definition can import other definitions. These imports are non-transitive, only the definitions defined in the imported document are available in the parent definition. This Document is mostly just a simple interface to the root definition.

After all definitions are loaded the definitions are resolved. This resolves references which were not yet available during the initial parsing phase.

Parameters

- **location** (*string*) – Location of this WSDL
- **transport** (*zeep.transports.Transport*) – The transport object to be used
- **base** (*str*) – The base location of this document
- **strict** (*bool*) – Indicates if strict mode is enabled

6.2.3 zeep.wsdl.definitions

A WSDL document exists out of a number of definitions. There are 6 major definitions, these are:

- types
- message
- portType
- binding
- port
- service

This module defines the definitions which occur within a WSDL document,

class zeep.wsdl.definitions.**AbstractMessage**(*name*)
Messages consist of one or more logical parts.

Each part is associated with a type from some type system using a message-typing attribute. The set of message-typing attributes is extensible. WSDL defines several such message-typing attributes for use with XSD:

- element: Refers to an XSD element using a QName.
- type: Refers to an XSD simpleType or complexType using a QName.

class zeep.wsdl.definitions.**AbstractOperation**(*name*, *input_message=None*, *output_message=None*, *fault_messages=None*, *parameter_order=None*, *wsa_action=None*)
Abstract operations are defined in the wsdl's portType elements.

class zeep.wsdl.definitions.**Binding**(*wsdl*, *name*, *port_name*)
Base class for the various bindings (SoapBinding / HttpBinding)

class zeep.wsdl.definitions.**MessagePart**(*element*, *type*)

element
Alias for field number 0

type
Alias for field number 1

class zeep.wsdl.definitions.**Operation**(*name*, *binding*)
Concrete operation

Contains references to the concrete messages

classmethod parse(*wsdl*, *xmlelement*, *binding*)
Definition:

```
<wsdl:operation name="nmtoken"> *
   <-- extensibility element (2) --> *
   <wsdl:input name="nmtoken"? > ?
       <-- extensibility element (3) -->
   </wsdl:input>
   <wsdl:output name="nmtoken"? > ?
       <-- extensibility element (4) --> *
   </wsdl:output>
   <wsdl:fault name="nmtoken"> *
       <-- extensibility element (5) --> *
   </wsdl:fault>
</wsdl:operation>
```

class zeep.wsdl.definitions.**Port**(*name*, *binding_name*, *xmlelement*)
Specifies an address for a binding, thus defining a single communication endpoint.

class zeep.wsdl.definitions.**Service**(*name*)
Used to aggregate a set of related ports.

6.2.4 zeep.wsdl.parse

zeep.wsdl.parse.**parse_abstract_message**(*wsdl: Definition*, *xmlelement: lxml.etree._Element*) → zeep.wsdl.definitions.AbstractMessage
Create an AbstractMessage object from a xml element.

Definition:

```
<definitions .... >
    <message name="nmtoken"> *
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </message>
</definitions>
```

Parameters

- **wsdl** – The parent definition instance
- **xmlelement** – The XML node

zeep.wsdl.parse.**parse_abstract_operation**(*wsdl: Definition*, *xmlelement: lxml.etree._Element*) → Optional[zeep.wsdl.definitions.AbstractOperation]
Create an AbstractOperation object from a xml element.

This is called from the parse_port_type function since the abstract operations are part of the port type element.

Definition:

```
<wsdl:operation name="nmtoken">*
   <wsdl:documentation .... /> ?
   <wsdl:input name="nmtoken"? message="qname">?
       <wsdl:documentation .... /> ?
   </wsdl:input>
   <wsdl:output name="nmtoken"? message="qname">?
       <wsdl:documentation .... /> ?
```

(continues on next page)

(continued from previous page)

```
   </wsdl:output>
   <wsdl:fault name="nmtoken" message="qname"> *
       <wsdl:documentation .... /> ?
   </wsdl:fault>
</wsdl:operation>
```

Parameters

- **wsdl** – The parent definition instance
- **xmlelement** – The XML node

zeep.wsdl.parse.**parse_port**(*wsdl: Definition*, *xmlelement: lxml.etree._Element*) → zeep.wsdl.definitions.Port

Create a Port object from a xml element.

This is called via the parse_service function since ports are part of the service xml elements.

Definition:

```
<wsdl:port name="nmtoken" binding="qname"> *
   <wsdl:documentation .... /> ?
   <-- extensibility element -->
</wsdl:port>
```

Parameters

- **wsdl** – The parent definition instance
- **xmlelement** – The XML node

zeep.wsdl.parse.**parse_port_type**(*wsdl: Definition*, *xmlelement: lxml.etree._Element*) → zeep.wsdl.definitions.PortType

Create a PortType object from a xml element.

Definition:

```
<wsdl:definitions .... >
    <wsdl:portType name="nmtoken">
        <wsdl:operation name="nmtoken" .... /> *
    </wsdl:portType>
</wsdl:definitions>
```

Parameters

- **wsdl** – The parent definition instance
- **xmlelement** – The XML node

zeep.wsdl.parse.**parse_service**(*wsdl: Definition*, *xmlelement: lxml.etree._Element*) → zeep.wsdl.definitions.Service

Definition:

```
<wsdl:service name="nmtoken"> *
    <wsdl:documentation .... />?
    <wsdl:port name="nmtoken" binding="qname"> *
       <wsdl:documentation .... /> ?
```

(continues on next page)

(continued from previous page)

```
        <-- extensibility element -->
    </wsdl:port>
    <-- extensibility element -->
</wsdl:service>
```

Example:

```
<service name="StockQuoteService">
  <documentation>My first service</documentation>
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
  </port>
</service>
```

Parameters

- **wsdl** – The parent definition instance
- **xmlelement** – The XML node

Basic implementation to support SOAP-Attachments

See https://www.w3.org/TR/SOAP-attachments

class zeep.wsdl.bindings.http.**HttpBinding**(*wsdl*, *name*, *port_name*)

class zeep.wsdl.bindings.http.**HttpGetBinding**(*wsdl*, *name*, *port_name*)

classmethod match(*node*)

Check if this binding instance should be used to parse the given node.

Parameters **node** (*lxml.etree._Element*) – The node to match against

send(*client*, *options*, *operation*, *args*, *kwargs*)

Called from the service

class zeep.wsdl.bindings.http.**HttpOperation**(*name*, *binding*, *location*)

classmethod parse(*definitions*, *xmlelement*, *binding*)

<wsdl:operation name="GetLastTradePrice"> <http:operation location="GetLastTradePrice"/>
<wsdl:input>

<mime:content type="application/x-www-form-urlencoded"/>

</wsdl:input> <wsdl:output>

<mime:mimeXml/>

</wsdl:output>

</wsdl:operation>

class zeep.wsdl.bindings.http.**HttpPostBinding**(*wsdl*, *name*, *port_name*)

classmethod match(*node*)

Check if this binding instance should be used to parse the given node.

Parameters **node** (*lxml.etree._Element*) – The node to match against

send(*client*, *options*, *operation*, *args*, *kwargs*)
Called from the service

class zeep.wsdl.bindings.soap.**Soap11Binding**(*wsdl*, *name*, *port_name*, *transport*, *default_style*)

class zeep.wsdl.bindings.soap.**Soap12Binding**(*wsdl*, *name*, *port_name*, *transport*, *default_style*)

class zeep.wsdl.bindings.soap.**SoapBinding**(*wsdl*, *name*, *port_name*, *transport*, *default_style*)

Soap 1.1/1.2 binding

classmethod match(*node*)
Check if this binding instance should be used to parse the given node.

Parameters **node** (*lxml.etree._Element*) – The node to match against

classmethod parse(*definitions*, *xmlelement*)
Definition:

```
<wsdl:binding name="nmtoken" type="qname"> *
    <-- extensibility element (1) --> *
    <wsdl:operation name="nmtoken"> *
       <-- extensibility element (2) --> *
       <wsdl:input name="nmtoken"? > ?
           <-- extensibility element (3) -->
       </wsdl:input>
       <wsdl:output name="nmtoken"? > ?
           <-- extensibility element (4) --> *
       </wsdl:output>
       <wsdl:fault name="nmtoken"> *
           <-- extensibility element (5) --> *
       </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
```

process_reply(*client*, *operation*, *response*)
Process the XML reply from the server.

Parameters

- **client** (*zeep.client.Client*) – The client with which the operation was called
- **operation** (zeep.wsdl.definitions.Operation) – The operation object from which this is a reply
- **response** (*requests.Response*) – The response object returned by the remote server

send(*client*, *options*, *operation*, *args*, *kwargs*)
Called from the service

Parameters

- **client** (*zeep.client.Client*) – The client with which the operation was called
- **options** (*dict*) – The binding options
- **operation** (zeep.wsdl.definitions.Operation) – The operation object from which this is a reply
- **args** (*tuple*) – The args to pass to the operation

- **kwargs** (*dict*) – The kwargs to pass to the operation

send_async(*client*, *options*, *operation*, *args*, *kwargs*)
Called from the async service

Parameters

- **client** (*zeep.client.Client*) – The client with which the operation was called
- **options** (*dict*) – The binding options
- **operation** (zeep.wsdl.definitions.Operation) – The operation object from which this is a reply
- **args** (*tuple*) – The args to pass to the operation
- **kwargs** (*dict*) – The kwargs to pass to the operation

class zeep.wsdl.bindings.soap.**SoapOperation**(*name*, *binding*, *nsmap*, *soapaction*, *style*)
Represent's an operation within a specific binding.

classmethod parse(*definitions*, *xmlelement*, *binding*, *nsmap*)
Definition:

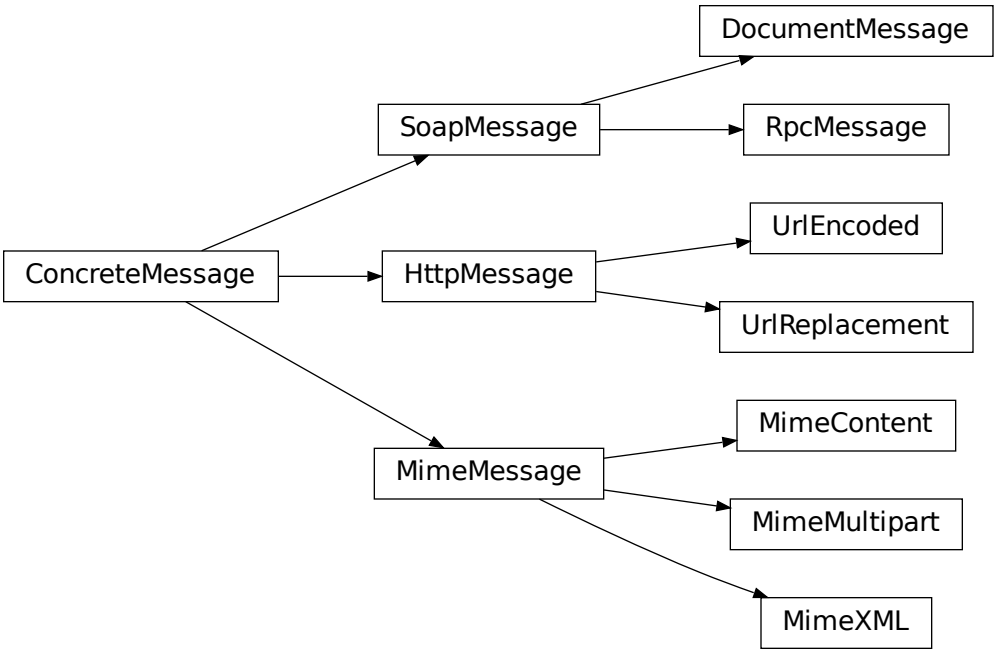
```
<wsdl:operation name="nmtoken"> *
    <soap:operation soapAction="uri"? style="rpc|document"?>?
    <wsdl:input name="nmtoken"? > ?
        <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="nmtoken"? > ?
        <-- extensibility element (4) --> *
   </wsdl:output>
   <wsdl:fault name="nmtoken"> *
        <-- extensibility element (5) --> *
   </wsdl:fault>
</wsdl:operation>
```

Example:

```
<wsdl:operation name="GetLastTradePrice">
  <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
  <wsdl:input>
    <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
  </wsdl:output>
  <wsdl:fault name="dataFault">
    <soap:fault name="dataFault" use="literal"/>
  </wsdl:fault>
</operation>
```

6.2.5 zeep.wsdl.messages

The messages are responsible for serializing and deserializing



6.2.6 zeep.wsdl.messages.base

class zeep.wsdl.messages.base.**ConcreteMessage**(*wsdl*, *name*, *operation*)
Represents the wsdl:binding -> wsdl:operation -> input/output node

class zeep.wsdl.messages.base.**SerializedMessage**(*path*, *headers*, *content*)

content
Alias for field number 2

headers
Alias for field number 1

path
Alias for field number 0

6.2.7 zeep.wsdl.messages.http

class zeep.wsdl.messages.http.**UrlEncoded**(*wsdl*, *name*, *operation*)
The urlEncoded element indicates that all the message parts are encoded into the HTTP request URI using the standard URI-encoding rules (name1=value&name2=value...).

The names of the parameters correspond to the names of the message parts. Each value contributed by the part is encoded using a name=value pair. This may be used with GET to specify URL encoding, or with POST to specify a FORM-POST. For GET, the "?" character is automatically appended as necessary.

class zeep.wsdl.messages.http.**UrlReplacement**(*wsdl*, *name*, *operation*)

The http:urlReplacement element indicates that all the message parts are encoded into the HTTP request URI using a replacement algorithm.

- The relative URI value of http:operation is searched for a set of search patterns.
- The search occurs before the value of the http:operation is combined with the value of the location attribute from http:address.
- There is one search pattern for each message part. The search pattern string is the name of the message part surrounded with parenthesis "(" and ")".
- For each match, the value of the corresponding message part is substituted for the match at the location of the match.
- Matches are performed before any values are replaced (replaced values do not trigger additional matches).

Message parts MUST NOT have repeating values. <http:urlReplacement/>

6.2.8 zeep.wsdl.messages.mime

class zeep.wsdl.messages.mime.**MimeContent**(*wsdl*, *name*, *operation*, *content_type*, *part_name*)

WSDL includes a way to bind abstract types to concrete messages in some MIME format.

Bindings for the following MIME types are defined:

- multipart/related
- text/xml
- application/x-www-form-urlencoded
- Others (by specifying the MIME type string)

The set of defined MIME types is both large and evolving, so it is not a goal for WSDL to exhaustively define XML grammar for each MIME type.

Parameters

- **wsdl** (zeep.wsdl.wsdl.Document) – The main wsdl document
- **name** –
- **operation** (zeep.wsdl.bindings.soap.SoapOperation) – The operation to which this message belongs
- **part_name** –

class zeep.wsdl.messages.mime.**MimeXML**(*wsdl*, *name*, *operation*, *part_name*)

To specify XML payloads that are not SOAP compliant (do not have a SOAP Envelope), but do have a particular schema, the mime:mimeXml element may be used to specify that concrete schema.

The part attribute refers to a message part defining the concrete schema of the root XML element. The part attribute MAY be omitted if the message has only a single part. The part references a concrete schema using the element attribute for simple parts or type attribute for composite parts

Parameters

- **wsdl** (zeep.wsdl.wsdl.Document) – The main wsdl document
- **name** –
- **operation** (zeep.wsdl.bindings.soap.SoapOperation) – The operation to which this message belongs

- **part_name** –

class zeep.wsdl.messages.mime.**MimeMultipart**(*wsdl*, *name*, *operation*, *part_name*)

The multipart/related MIME type aggregates an arbitrary set of MIME formatted parts into one message using the MIME type "multipart/related".

The mime:multipartRelated element describes the concrete format of such a message:

```
<mime:multipartRelated>
    <mime:part> *
        <-- mime element -->
    </mime:part>
</mime:multipartRelated>
```

The mime:part element describes each part of a multipart/related message. MIME elements appear within mime:part to specify the concrete MIME type for the part. If more than one MIME element appears inside a mime:part, they are alternatives.

Parameters

- **wsdl** (zeep.wsdl.wsdl.Document) – The main wsdl document
- **name** –
- **operation** (zeep.wsdl.bindings.soap.SoapOperation) – The operation to which this message belongs
- **part_name** –

6.2.9 zeep.wsdl.messages.soap

class zeep.wsdl.messages.soap.**DocumentMessage**(**args*, ***kwargs*)

In the document message there are no additional wrappers, and the message parts appear directly under the SOAP Body element.

Parameters

- **wsdl** (*zeep.wsdl.Document*) – The main wsdl document
- **name** –
- **operation** (zeep.wsdl.bindings.soap.SoapOperation) – The operation to which this message belongs
- **type** (*str*) – 'input' or 'output'
- **nsmap** (*dict*) – The namespace mapping

deserialize(*envelope*)

Deserialize the SOAP:Envelope and return a CompoundValue with the result.

classmethod parse(*definitions*, *xmlelement*, *operation*, *type*, *nsmap*)
Parse a wsdl:binding/wsdl:operation/wsdl:operation for the SOAP implementation.

Each wsdl:operation can contain three child nodes:

- input
- output
- fault

Definition for input/output:

```
<input>
  <soap:body parts="nmtokens"? use="literal|encoded"
             encodingStyle="uri-list"? namespace="uri"?>

  <soap:header message="qname" part="nmtoken" use="literal|encoded"
               encodingStyle="uri-list"? namespace="uri"?>*
    <soap:headerfault message="qname" part="nmtoken"
                      use="literal|encoded"
                      encodingStyle="uri-list"? namespace="uri"?/>*
  </soap:header>
</input>
```

And the definition for fault:

```
<soap:fault name="nmtoken" use="literal|encoded"
            encodingStyle="uri-list"? namespace="uri"?>
```

resolve(*definitions*, *abstract_message*)
Resolve the data in the self._resolve_info dict (set via parse())

This creates three xsd.Element objects:

- self.header
- self.body
- self.envelope (combination of headers and body)

XXX headerfaults are not implemented yet.

serialize(**args*, ***kwargs*)
Create a SerializedMessage for this message

class zeep.wsdl.messages.soap.**RpcMessage**(*wsdl*, *name*, *operation*, *type*, *nsmap*)
In RPC messages each part is a parameter or a return value and appears inside a wrapper element within the body.

The wrapper element is named identically to the operation name and its namespace is the value of the namespace attribute. Each message part (parameter) appears under the wrapper, represented by an accessor named identically to the corresponding parameter of the call. Parts are arranged in the same order as the parameters of the call.

Parameters

- **wsdl** (*zeep.wsdl.Document*) – The main wsdl document
- **name** –
- **operation** (zeep.wsdl.bindings.soap.SoapOperation) – The operation to which this message belongs
- **type** (*str*) – 'input' or 'output'
- **nsmap** (*dict*) – The namespace mapping

deserialize(*envelope*)

Deserialize the SOAP:Envelope and return a CompoundValue with the result.

classmethod parse(*definitions*, *xmlelement*, *operation*, *type*, *nsmap*)

Parse a wsdl:binding/wsdl:operation/wsdl:operation for the SOAP implementation.

Each wsdl:operation can contain three child nodes:

- input
- output
- fault

Definition for input/output:

```
<input>
  <soap:body parts="nmtokens"? use="literal|encoded"
             encodingStyle="uri-list"? namespace="uri"?>

  <soap:header message="qname" part="nmtoken" use="literal|encoded"
               encodingStyle="uri-list"? namespace="uri"?>*
    <soap:headerfault message="qname" part="nmtoken"
                      use="literal|encoded"
                      encodingStyle="uri-list"? namespace="uri"?/>*
  </soap:header>
</input>
```

And the definition for fault:

```
<soap:fault name="nmtoken" use="literal|encoded"
            encodingStyle="uri-list"? namespace="uri"?>
```

resolve(*definitions*, *abstract_message*)

Resolve the data in the self._resolve_info dict (set via parse())

This creates three xsd.Element objects:

- self.header
- self.body

- self.envelope (combination of headers and body)

XXX headerfaults are not implemented yet.

serialize(**args*, ***kwargs*)
Create a SerializedMessage for this message

6.2.10 zeep.wsdl.utils

6.2.11 zeep.xsd

class zeep.xsd.schema.**Schema**(*node=None*, *transport=None*, *location=None*, *settings=None*)
A schema is a collection of schema documents.

create_new_document(*node*, *url*, *base_url=None*, *target_namespace=None*)

Return type *zeep.xsd.schema.SchemaDocument*

elements
Yield all globla xsd.Type objects

Return type Iterable of zeep.xsd.Element

get_attribute(*qname*) → zeep.xsd.elements.attribute.Attribute
Return a global xsd.attribute object with the given qname

get_attribute_group(*qname*) → zeep.xsd.elements.attribute.AttributeGroup
Return a global xsd.attributeGroup object with the given qname

get_element(*qname*) → zeep.xsd.elements.element.Element
Return a global xsd.Element object with the given qname

get_group(*qname*) → zeep.xsd.elements.indicators.Group
Return a global xsd.Group object with the given qname.

get_type(*qname*, *fail_silently=False*)
Return a global xsd.Type object with the given qname

Return type zeep.xsd.ComplexType or zeep.xsd.AnySimpleType

is_empty
Boolean to indicate if this schema contains any types or elements

merge(*schema*)
Merge an other XSD schema in this one

types
Yield all global xsd.Type objects

Return type Iterable of zeep.xsd.ComplexType

class zeep.xsd.schema.**SchemaDocument**(*namespace*, *location*, *base_url*)
A Schema Document consists of a set of schema components for a specific target namespace.

This represents an xsd:Schema object

get_attribute(*qname*) → zeep.xsd.elements.attribute.Attribute
Return a xsd.Attribute object from this schema

get_attribute_group(*qname*) → zeep.xsd.elements.attribute.AttributeGroup
Return a xsd.AttributeGroup object from this schema

get_element(*qname*) → zeep.xsd.elements.element.Element
Return a xsd.Element object from this schema

get_group(*qname*) → zeep.xsd.elements.indicators.Group
Return a xsd.Group object from this schema

get_type(*qname: lxml.etree.QName*)
Return a xsd.Type object from this schema

Return type zeep.xsd.ComplexType or zeep.xsd.AnySimpleType

load(*schema*, *node*)
Load the XML Schema passed in via the node attribute.

register_attribute(*qname: str*, *value: zeep.xsd.elements.attribute.Attribute*)
Register a xsd:Attribute in this schema

register_attribute_group(*qname: lxml.etree.QName*, *value: zeep.xsd.elements.attribute.AttributeGroup*) → None
Register a xsd:AttributeGroup in this schema

register_element(*qname: lxml.etree.QName*, *value: zeep.xsd.elements.element.Element*)
Register a xsd.Element in this schema

register_group(*qname: lxml.etree.QName*, *value: zeep.xsd.elements.indicators.Group*)
Register a xsd:Group in this schema

register_import(*namespace*, *schema*)
Register an import for an other schema document.

register_type(*qname: lxml.etree.QName*, *value: zeep.xsd.types.base.Type*)
Register a xsd.Type in this schema

zeep.xsd.utils.**create_prefixed_name**(*qname*, *schema*)
Convert a QName to a xsd:name ('ns1:myType').

Return type str

class zeep.xsd.valueobjects.**AnyObject**(*xsd_object*, *value*)
Create an any object

Parameters

- **xsd_object** – the xsd type
- **value** – The value

class zeep.xsd.valueobjects.**CompoundValue**(**args*, ***kwargs*)
Represents a data object for a specific xsd:complexType.

class zeep.xsd.visitor.**SchemaVisitor**(*schema*, *document*)
Visitor which processes XSD files and registers global elements and types in the given schema.

Notes:

TODO: include and import statements can reference other nodes. We need to load these first. Always global.

Parameters

- **schema** (zeep.xsd.schema.Schema) –
- **document** (zeep.xsd.schema.SchemaDocument) –

visit_all(*node*, *parent*)
Allows the elements in the group to appear (or not appear) in any order in the containing element.

Definition:

```
<all
  id = ID
  maxOccurs= 1: 1
  minOccurs= (0 | 1): 1
  {any attributes with non-schema Namespace...}>
Content: (annotation?, element*)
</all>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_annotation(*node*, *parent*)
Defines an annotation.

Definition:

```
<annotation
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (appinfo | documentation)*
</annotation>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_any(*node*, *parent*)
Definition:

```
<any
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  namespace = "(##any | ##other) |
    List of (anyURI | (##targetNamespace |  ##local))) : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema Namespace...}>
Content: (annotation?)
</any>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_any_attribute(*node*, *parent*)
Definition:

```
<anyAttribute
  id = ID
  namespace = ((##any | ##other) |
    List of (anyURI | (##targetNamespace | ##local))) : ##any
  processContents = (lax | skip | strict): strict
  {any attributes with non-schema Namespace...}>
Content: (annotation?)
</anyAttribute>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_attribute(*node: lxml.etree._Element*, *parent: lxml.etree._Element*) → Union[zeep.xsd.elements.attribute.Attribute, zeep.xsd.elements.references.RefAttribute]

Declares an attribute.

Definition:

```
<attribute
  default = string
  fixed = string
  form = (qualified | unqualified)
  id = ID
  name = NCName
  ref = QName
  type = QName
  use = (optional | prohibited | required): optional
  {any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleType?))
</attribute>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_attribute_group(*node*, *parent*)

Definition:

```
<attributeGroup
  id = ID
  name = NCName
  ref = QName
  {any attributes with non-schema Namespace...}>
Content: (annotation?),
         ((attribute | attributeGroup)*, anyAttribute?))
</attributeGroup>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_choice(*node*, *parent*)

Definition:

```
<choice
  id = ID
  maxOccurs= (nonNegativeInteger | unbounded) : 1
  minOccurs= nonNegativeInteger : 1
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (element | group | choice | sequence | any)*)
</choice>
```

visit_complex_content(*node*, *parent*)

The complexContent element defines extensions or restrictions on a complex type that contains mixed content or elements only.

Definition:

```
<complexContent
  id = ID
  mixed = Boolean
  {any attributes with non-schema Namespace}...>
Content: (annotation?,  (restriction | extension))
</complexContent>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_complex_type(*node*, *parent*)

Definition:

```
<complexType
  abstract = Boolean : false
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
  id = ID
  mixed = Boolean : false
  name = NCName
  {any attributes with non-schema Namespace...}>
Content: (annotation?, (simpleContent | complexContent |
          ((group | all | choice | sequence)?,
          ((attribute | attributeGroup)*, anyAttribute?))))
</complexType>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_element(*node*, *parent*)

Definition:

```
<element
  abstract = Boolean : false
  block = (#all | List of (extension | restriction | substitution))
```

(continues on next page)

(continued from previous page)

```
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = Boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (
          (simpleType | complexType)?, (unique | key | keyref)*))
</element>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_extension_complex_content(*node*, *parent*)

Definition:

```
<extension
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (
            (group | all | choice | sequence)?,
            ((attribute | attributeGroup)*, anyAttribute?)))
</extension>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_extension_simple_content(*node*, *parent*)

Definition:

```
<extension
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?))
</extension>
```

visit_group(*node*, *parent*)

Groups a set of element declarations so that they can be incorporated as a group into complex type definitions.

Definition:

```
<group
  name= NCName
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  ref = QName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (all | choice | sequence))
</group>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_import(*node*, *parent*)

Definition:

```
<import
  id = ID
  namespace = anyURI
  schemaLocation = anyURI
  {any attributes with non-schema Namespace}...>
Content: (annotation?)
</import>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_include(*node*, *parent*)

Definition:

```
<include
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema Namespace}...>
Content: (annotation?)
</include>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_list(*node*, *parent*)

Definition:

```
<list
  id = ID
  itemType = QName
  {any attributes with non-schema Namespace}...>
```

(continues on next page)

(continued from previous page)

```
Content: (annotation?, (simpleType?))
</list>
```

The use of the simpleType element child and the itemType attribute is mutually exclusive.

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_notation(*node*, *parent*)

Contains the definition of a notation to describe the format of non-XML data within an XML document. An XML Schema notation declaration is a reconstruction of XML 1.0 NOTATION declarations.

Definition:

```
<notation
  id = ID
  name = NCName
  public = Public identifier per ISO 8879
  system = anyURI
  {any attributes with non-schema Namespace}...>
Content: (annotation?)
</notation>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_restriction_complex_content(*node*, *parent*)

Definition:

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (group | all | choice | sequence)?,
        ((attribute | attributeGroup)*, anyAttribute?))
</restriction>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_restriction_simple_content(*node*, *parent*)

Definition:

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?,
    (simpleType?, (
```

(continues on next page)

(continued from previous page)

```
        minExclusive | minInclusive | maxExclusive | maxInclusive |
        totalDigits |fractionDigits | length | minLength |
        maxLength | enumeration | whiteSpace | pattern)*
    )?, ((attribute | attributeGroup)*, anyAttribute?))
</restriction>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_restriction_simple_type(*node*, *parent*)

Definition:

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?,
    (simpleType?, (
        minExclusive | minInclusive | maxExclusive | maxInclusive |
        totalDigits |fractionDigits | length | minLength |
        maxLength | enumeration | whiteSpace | pattern)*))
</restriction>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_schema(*node*)

Visit the xsd:schema element and process all the child elements

Definition:

```
<schema
  attributeFormDefault = (qualified | unqualified): unqualified
  blockDefault = (#all | List of (extension | restriction | substitution) : ''
  elementFormDefault = (qualified | unqualified): unqualified
  finalDefault = (#all | List of (extension | restriction | list | union): ''
  id = ID
  targetNamespace = anyURI
  version = token
  xml:lang = language
  {any attributes with non-schema Namespace}...>
Content: (
    (include | import | redefine | annotation)*,
    (((simpleType | complexType | group | attributeGroup) |
      element | attribute | notation),
     annotation*)*)
</schema>
```

Parameters **node** (*lxml.etree._Element*) – The XML node

visit_sequence(*node*, *parent*)

Definition:

```
<sequence
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema Namespace}...>
Content: (annotation?,
          (element | group | choice | sequence | any)*)
</sequence>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_simple_content(*node*, *parent*)

Contains extensions or restrictions on a complexType element with character data or a simpleType element as content and contains no elements.

Definition:

```
<simpleContent
  id = ID
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (restriction | extension))
</simpleContent>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_simple_type(*node*, *parent*)

Definition:

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = NCName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (restriction | list | union))
</simpleType>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_union(*node*, *parent*)

Defines a collection of multiple simpleType definitions.

Definition:

```
<union
  id = ID
  memberTypes = List of QNames
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (simpleType*))
</union>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

visit_unique(*node*, *parent*)

Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope. The value must be unique or nil.

Definition:

```
<unique
  id = ID
  name = NCName
  {any attributes with non-schema Namespace}...>
Content: (annotation?, (selector, field+))
</unique>
```

Parameters

- **node** (*lxml.etree._Element*) – The XML node
- **parent** (*lxml.etree._Element*) – The parent XML node

CHAPTER 7

Changelog

7.1 Changelog

7.1.1 4.1.0 (2021-08-15)

- Remove last dependency on *six* (#1250)
- Use *platformdirs* instead of the *appsdirs* dependency (#1244)
- Pass digest method when signing timestamp node(#1201)
- Fix settings context manager when an exception is raised (#1193)
- Don't render decimals using scientific notation (#1191)
- Remove dependency on *defusedxml* (deprecated) (#1179)
- Improve handling of str values for Duration (#1165)

7.1.2 4.0.0 (2020-10-12)

- Drop support for Python 2.7, 3.3, 3.4 and 3.5
- Remove both the aiohttp and the tornado transport class from Zeep. These can be moved to their own Python package if anyone is interested.
- Add *zeep.transports.AsyncTransport* which is based on *httpx*. Note that loading wsdl files is still a sync process but operations can be executed via async.
- Start adding more typehints to the package

7.1.3 3.4.0 (2019-06-05)

- Allow passing xsd.Nil to sequences (#959, smilingDima)

- Add support for passing custom digest and signature methods during WSSE signing (#957, Florian Apolloner)
- Fix path resolving during XSD imports (#962, adambogocz)

7.1.4 3.3.1 (2019-03-10)

- Fix issue with empty xsd:import statements on Python 2.7 (#930)

7.1.5 3.3.0 (2019-03-08)

- Extend the force_https flag to also force loading xsd files from https when a http url is encountered from a https domain
- Fix handling recursive xsd imports when the url's are enforced from http to https.
- Fix reference attribute when using the Binary Security Token (#873, tpazderka)
- Add support for the WSAM namespace (#890, David Adam)

7.1.6 3.2.0 (2018-12-17)

- Fix abstract message check for NoneType before attempting to access parts
- Improve support for 'Chameleon' XSD schemas (#879, #888)
- Fix resolving qualified references (#879)
- Fix issue with duplicate soap:body tags when multiple parts used (#879)
- Fix Choice with unbound Any element (#871)
- Add xsd_ignore_sequence_order flag (#880)
- Add support for timestamp token in WSS headers (#817)
- Accept strings for xsd.DateTime (#886)

7.1.7 3.1.0 (2018-07-28)

- Fix SSL issue on with *TornadoAsyncTransport* (#792)
- Fix passing strict keyword in XML loader (#787)
- Update documentation

7.1.8 3.0.0 (2018-06-16)

This is a major release, and contains a number of backwards incompatible changes to the API.

- Refactor the settings logic in Zeep. All settings are now configured using the zeep.settings.Settings() class.
- Allow control of defusedxml settings via zeep.Settings (see #567, #391)
- Add ability to set specific http headers for each call (#758)
- Skip the xsd:annotation element in the all:element (#737)
- Add Settings.force_https as option so that it can be disabled (#605)

- Strip spaces from QName's when parsing xsd's (#719)
- Fix DateTime parsing when only a date is returned (#672)
- Fix handling of nested optional any elements (#556)
- Check if part exists before trying to delete it (#659)

7.1.9 2.5.0 (2018-01-06)

- Fix AnyType value rendering by guessing the xsd type for the value (#552)
- Fix AnySimpleType.xmlvalue() not implemented exception (#644)
- Add __dir__ method to value objects returned by Zeep
- Don't require content for 201 and 202 status codes (#613)
- Fix wheel package by cleaning the build directory correctly (#634)
- Handle Nil values on complexType with SimpleContent elements (#604)
- Add Client.namespaces method to list all namespaces available
- Improve support for auto-completion (#537)

7.1.10 2.4.0 (2017-08-26)

- Add support for tornado async transport via gen.coroutine (#530, Kateryna Burda)
- Check if soap:address is defined in the service port instead of raising an exception (#527)
- Update packaging (stop using find_packages()) (#529)
- Properly handle None values when rendering complex types (#526)
- Fix generating signature for empty wsdl messages (#542)
- Support passing strings to xsd:Time objects (#540)

7.1.11 2.3.0 (2017-08-06)

- The XML send to the server is no longer using `pretty_print=True` (#484)
- Refactor of the multiref support to fix issues with child elements (#489)
- Add workaround to support negative durations (#486)
- Fix creating XML documents for operations without aguments (#479)
- Fix xsd:extension on xsd:group elements (#523)

7.1.12 2.2.0 (2017-06-19)

- Automatically import the soap-encoding schema if it is required (#473)
- Add support for XOP messages (this is a rewrite of #325 by vashek)

7.1.13 2.1.1 (2017-06-11)

- Fix previous release, it contained an incorrect dependency (Mock 2.1.) due to bumpversion :-(

7.1.14 2.1.0 (2017-06-11)

- Fix recursion error while creating the signature for a global element when it references itself (via ref attribute).
- Update Client.create_message() to apply plugins and wsse (#465)
- Fix handling unknown xsi types when parsing elements using xsd:anyType (#455)

7.1.15 2.0.0 (2017-05-22)

This is a major release, and contains a number of backwards incompatible changes to the API.

- Default values of optional elements are not set by default anymore (#423)
- Refactor the implementation of wsdl:arrayType too make the API more pythonic (backwards incompatible).
- The call signature for Client.create_message() was changed. It now requires the service argument:

```
Client.create_message(service, operation_name, *args, **kwargs)
```

- Choice elements now only work with keyword arguments and raise an exception if positional arguments are passed (#439)
- Implement initial multiref support for SOAP RPC (#326). This was done using really good real-world tests from vstoykov (thanks!)
- Fix exception on empty SOAP response (#442)
- Fix XSD default values for boolean types (Bartek Wójcicki, #386)

7.1.16 1.6.0 (2017-04-27)

- Implement ValueObject.__json__ for json serialization (#258)
- Improve handling of unexpected elements for soap:header (#378)
- Accept unexpected elements in complexTypes when strict is False
- Fix elementFormDefault/attributeFormDefault for xsd:includes (#426)

7.1.17 1.5.0 (2017-04-22)

- Fix issue where values of indicators (sequence/choice/all) would write to the same internal dict. (#425)
- Set default XML parse mode to strict as was intended (#332)
- Add support for pickling value objects (#417)
- Add explicit Nil value via `zeep.xsd.Nil` (#424)
- Add xml_huge_tree kwarg to the Client() to enable lxml's huge_tree mode, this is disabled by default (#332)
- Add support to pass base-types to type extensions (#416)
- Handle wsdl errors more gracefully by disabling the invalid operations instead of raising an exception (#407, #387)

7.1.18 1.4.1 (2017-04-01)

- The previous release (1.4.0) contained an incorrect dependency due to bumpversion moving all 1.3.0 versions to 1.4.0. This fixes it.

7.1.19 1.4.0 (2017-04-01)

- Hardcode the xml prefix to the xml namespace as defined in the specs (#367)
- Fix parsing of unbound sequences within xsd choices (#380)
- Use logger.debug() for debug related logging (#369)
- Add the `Client.raw_response` option to let zeep return the raw transport response (requests.Response) instead of trying to parse it.
- Handle minOccurs/maxOccurs properlhy for xsd:Group elements. This also fixes a bug in the xsd:Choice handling for multiple elements (#374, #410)
- Fix raising XMLSyntaxError when loading invalid XML (Antanas Sinica, #396)

7.1.20 1.3.0 (2017-03-14)

- Add support for nested xsd:choice elements (#370)
- Fix unresolved elements for xsd:extension, this was a regression introduced in 1.2.0 (#377)

7.1.21 1.2.0 (2017-03-12)

- Add flag to disable strict mode in the Client. This allows zeep to better work with non standard compliant SOAP Servers. See the documentation for usage and potential downsides.
- Minor refactor of resolving of elements for improved performance
- Support the SOAP 1.2 'http://www.w3.org/2003/05/soap/bindings/HTTP/' transport uri (#355)
- Fallback to matching wsdl lookups to matching when the target namespace is empty (#356)
- Improve the handling of xsd:includes, the default namespace of the parent schema is now also used during resolving of the included schema. (#360)
- Properly propagate the global flag for complex types when an xsd:restriction is used (#360)
- Filter out duplicate types and elements when dump the wsdl schema (#360)
- Add `zeep.CachingClient()` which enables the SqliteCache by default

7.1.22 1.1.0 (2017-02-18)

- Fix an attribute error when an complexType used xsd:anyType as base restriction (#352)
- Update asyncio transport to return requests.Response objects (#335)

7.1.23 1.0.0 (2017-01-31)

- Use cgi.parse_header() to extract media_type for multipart/related checks (#327)
- Don't ignore nil elements, instead return None when parsing xml (#328)
- Fix regression when using WSA with an older lxml version (#197)

7.1.24 0.27.0 (2017-01-28)

- Add support for SOAP attachments (multipart responses). (Dave Wapstra, #302)
- Update xsd:anyType to return the xml elements when no type is given via the xsi:type attribute (#284)
- Fix parsing Any elements when a restriction is used (soap-enc:array) (#322)

7.1.25 0.26.0 (2017-01-26)

This release again introduces some backwords incompatibilties. The next release will hopefully be 1.0 which will introduce semver.

- **backwards-incompatible**: The Transport class now accepts a `requests.Session()` object instead of `http_auth` and `verify`. This allows for more flexibility.
- **backwards-incompatible**: Zeep no longer sets a default cache backend. Please see http://docs.python-zeep.org/en/master/transport.html#caching for information about how to configure a cache.
- Add `zeep.xsd.SkipValue` which instructs the serialize to ignore the element.
- Support duplicate target namespaces in the wsdl definition (#320)
- Fix resolving element/types for xsd schema's with duplicate tns (#319)

7.1.26 0.25.0 (2017-01-23)

- **Important:** Add basic validation against the xsd. It currently will only validate the minOccurs/maxOccurs but this will be extended in the future.
- Add support for duplicate namespace definitions. Previously imports for namespaces which were already imported were ignored. It will now search through all matching schemas with the tns to find a specific object (#204)
- Fix xsd:extension for sequence -> choice. (#257)
- Improve serializing attributes when the values were passed as a dict (#125)

7.1.27 0.24.0 (2016-12-16)

- Don't fail the parsing of responses if an xsi:type references an non-existing type. Instead log a message (#273)
- Fix serializing etree.Element instances in the helpers.serialize function (#255)
- Fix infinite loop during parsing of xsd.Sequence where max_occurs is unbounded (#256)
- Make the xsd.Element name kwarg required
- Improve handling of the xsd:anyType element when passed instances of complexType's (#252)
- Silently ignore unsupported binding transports instead of an hard error (#277)

- Support microseconds for xsd.dateTime and xsd.Time (#280)
- Don't mutate passed values to the zeep operations (#280)

7.1.28 0.23.0 (2016-11-24)

- Add Client.set_default_soapheaders() to set soapheaders which are to be used on all operations done via the client object.
- Add basic support for asyncio using aiohttp. Many thanks to chrisimcevoy for the initial implementation! Please see https://github.com/mvantellingen/python-zeep/pull/207 and https://github.com/mvantellingen/python-zeep/pull/251 for more information
- Fix recursion error when generating the call signature (jaceksnet, #264)

7.1.29 0.22.1 (2016-11-22)

- Fix reversed() error (jaceksnet) (#260)
- Better error message when unexpected xml elements are encountered in sequences.

7.1.30 0.22.0 (2016-11-13)

- Force the soap:address / http:address to HTTPS when the wsdl is loaded from a https url (#228)
- Improvements to the xsd:union handling. The matching base class is now used for serializing/deserializing the values. If there is no matching base class then the raw value is returned. (#195)
- Fix handling of xsd:any with maxOccurs > 1 in xsd:choice elements (#253)
- Add workaround for schema's importing the xsd from http://www.w3.org/XML/1998/namespace (#220)
- Add new Client.type_factory(namespace) method which returns a factory to simplify creation of types.

7.1.31 0.21.0 (2016-11-02)

- Don't error on empty xml namespaces declarations in inline schema's (#186)
- Wrap importing of sqlite3 in try..except for Google App Engine (#243)
- Don't use pkg_resources to determine the zeep version, use __version__ instead (#243).
- Fix SOAP arrays by wrapping children in the appropriate element (joeribekker, #236)
- Add `operation_timeout` kwarg to the Transport class to set timeouts for operations. The default is still no timeout (#140)
- Introduce client.options context manager to temporarily override various options (only timeout for now) (#140)
- Wrap the parsing of xml values in a try..except block and log an error instead of throwing an exception (#137)
- Fix xsd:choice xml rendering with nested choice/sequence structure (#221)
- Correctly resolve header elements of which the message part defines the type instead of element. (#199)

7.1.32 0.20.0 (2016-10-24)

- Major performance improvements / lower memory usage. Zeep now no longer copies data and alters it in place but instead uses a set to keep track of modified data.
- Fix parsing empty soap response (#223)
- Major refactor of the xsd:extension / xsd:restriction implementation.
- Better support for xsd:anyType, by re-using the xsd.AnyObject (#229)
- Deserialize SOAP response without message elements correctly (#237)

7.1.33 0.19.0 (2016-10-18)

- **backwards-incompatible**: If the WSDL defines that the endpoint returns soap:header elements and/or multple soap:body messages then the return signature of the operation is changed. You can now explcitly access the body and header elements.
- Fix parsing HTTP bindings when there are no message elements (#185)
- Fix deserializing RPC responses (#219
- Add support for SOAP 1.2 Fault subcodes (#210, vashek)
- Don't alter the _soapheaders elements during rendering, instead create a deepcopy first. (#188)
- Add the SOAPAction to the Content-Type header in SOAP 1.2 bindings (#211)
- Fix issue when mixing elements and any elements in a choice type (#192)
- Improving parsing of results for union types (#192)
- Make ws-addressing work with lxml < 3.5 (#209)
- Fix recursion error when xsi:type='anyType' is given. (#198)

7.1.34 0.18.1 (2016-09-23)

- PyPi release error

7.1.35 0.18.0 (2016-09-23)

- Fix parsing Any elements by using the namespace map of the response node instead of the namespace map of the wsdl. (#184, #164)
- Improve handling of nested choice elements (choice>sequence>choice)

7.1.36 0.17.0 (2016-09-12)

- Add support for xsd:notation (#183)
- Add improvements to resolving phase so that all objects are resolved.
- Improve implementation of xsd.attributeGroup and xsd.UniqueType
- Create a deepcopy of the args and kwargs passed to objects so that the original are unmodified.
- Improve handling of wsdl:arrayType

7.1.37 0.16.0 (2016-09-06)

- Fix error when rendering choice elements with have sequences as children, see #150
- Re-use credentials passed to python -mzeep <wsdl> (#130)
- Workaround invalid usage of qualified vs non-qualified element tags in the response handling (#176)
- Fix regression when importing xsd:schema's via wsdl:import statements (#179)

7.1.38 0.15.0 (2016-09-04)

- All wsdl documents and xsd schemas are now globally available for eachother. While this is not correct according to the (messy) soap specifications, it does make zeep more compatible with all the invalid wsdl documents out there. (#159)
- Implement support for attributeGroup (#160)
- Add experimental support for ws-addressing (#92)
- Fix handling of Mime messages with no parts (#168)
- Workaround an issue where soap servers don't qualify references (#170)
- Correctly process attributes which are passed as a dictionary. (#125)
- Add support for plugins, see documentation for examples.
- Fix helpers.serialize_object for lists of objects (#123).
- Add HistoryPlugin which ofers last_sent and last_received properties (#93).

7.1.39 0.14.0 (2016-08-03)

- Global attributes are now always correctly handled as qualified. (#129)
- Fix parsing xml data containing simpleContent types (#136).
- Set xsi:nil attribute when serializing objects to xml (#141)
- Fix rendering choice elements when the element is mixed with other elements in a sequence (#150)
- Fix maximum recursion error for recursive xsd:include elements
- Make wsdl:import statements transitive. (#149)
- Merge xsd:schema's which are spread around imported wsdl objects. (#146)
- Don't raise exception when no value is given for AnyAttribute (#152)

7.1.40 0.13.0 (2016-07-17)

- Use warnings.warn() for duplicate target namespaces instead of raising an exception. This better matches with what lxml does.
- **backwards-incompatible**: The `persistent` kwarg is removed from the SqliteCache.__init__() call. Use the new InMemoryCache() instead when you don't want to persist data. This was required to make the SqliteCache backend thread-safe since we now open/close the db when writing/reading from it (with an additional lock).
- Fix zeep.helpers.serialize_object() for nested objects (#123)

- Remove fallback between soap 1.1 and soap 1.2 namespaces during the parsing of the wsdl. This should not be required.

7.1.41 0.12.0 (2016-07-09)

- **backwards-incompatible**: Choice elements are now unwrapped if maxOccurs=1. This results in easier operation definitions when choices are used.
- **backwards-incompatible**: The _soapheader kwarg is renamed to _soapheaders and now requires a nested dictionary with the header name as key or a list of values (value object or lxml.etree.Element object). Please see the call signature of the function using `python -mzeep <wsdl>`.
- Support the element ref's to xsd:schema elements.
- Improve the signature() output of element and type definitions
- Accept lxml.etree.Element objects as value for Any elements.
- And various other fixes

7.1.42 0.11.0 (2016-07-03)

- **backwards-incompatible**: The kwarg name for Any and Choice elements are renamed to generic `_value_N` names.
- **backwards-incompatible**: Client.set_address() is replaced with the Client.create_service() call
- Auto-load the http://schemas.xmlsoap.org/soap/encoding/ schema if it is referenced but not imported. Too many XSD's assume that the schema is always available.
- Major refactoring of the XSD handling to correctly support nested xsd:sequence elements.
- Add `logger.debug()` calls around Transport.post() to allow capturing the content send/received from the server
- Add proper support for default values on attributes and elements.

7.1.43 0.10.0 (2016-06-22)

- Make global elements / types truly global by refactoring the Schema parsing. Previously the lookups where non-transitive, but this should only be the case during parsing of the xml schema.
- Properly unwrap XML responses in soap.DocumentMessage when a choice is the root element. (#80)
- Update exceptions structure, all zeep exceptions are now using zeep.exceptions.Error() as base class.

7.1.44 0.9.1 (2016-06-17)

- Quote the SOAPAction header value (Derek Harland)
- Undo fallback for SOAPAction if it is empty (#83)

7.1.45 0.9.0 (2016-06-14)

- Use the appdirs module to retrieve the OS cache path. Note that this results in an other default cache path then previous releases! See https://github.com/ActiveState/appdirs for more information.
- Fix regression when initializing soap objects with invalid kwargs.
- Update wsse.UsernameToken to set encoding type on nonce (Antonio Cuni)
- Remove assert statement in soap error handling (Eric Waller)
- Add '–no-verify' to the command line interface. (#63)
- Correctly xsi:type attributes on unbounded elements. (nicholjy) (#68)
- Re-implement xsd:list handling
- Refactor logic to open files from filesystem.
- Refactor the xsd:choice implementation (serializing/deserializing)
- Implement parsing of xsd:any elements.

7.1.46 0.8.1 (2016-06-08)

- Use the operation name for the xml element which wraps the parameters in for soap RPC messages (#60)

7.1.47 0.8.0 (2016-06-07)

- Add ability to override the soap endpoint via *Client.set_address()*
- Fix parsing ComplexTypes which have no child elements (#50)
- Handle xsi:type attributes on anyType's correctly when deserializing responses (#17)
- Fix xsd:restriction on xsd:simpleType's when the base type wasn't defined yet. (#59)
- Add xml declaration to the generate xml strings (#60)
- Fix xsd:import statements without schemaLocation (#58)

7.1.48 0.7.1 (2016-06-01)

- Fix regression with handling wsdl:import statements for messages (#47)

7.1.49 0.7.0 (2016-05-31)

- Add support HTTP authentication (mcordes). This adds a new attribute to the Transport client() which passes the http_auth value to requests. (#31)
- Fix issue where setting cache=None to Transport class didn't disable caching.
- Refactor handling of wsdl:imports, don't merge definitions but instead lookup values in child definitions. (#40)
- Remove unused namespace declarations from the generated SOAP messages.
- Update requirement of six>=1.0.0 to six>=1.9.0 (#39)
- Fix handling of xsd:choice, xsd:group and xsd:attribute (#30)

- Improve error messages
- Fix generating soap messages when sub types are used via xsd extensions (#36)
- Improve handling of custom soap headers (#33)

7.1.50 0.6.0 (2016-05-21)

- Add missing *name* attributes to xsd.QName and xsd.NOTATION (#15)
- Various fixes related to the Choice element
- Support xsd:include
- Experimental support for HTTP bindings
- Removed *Client.get_port()*, use *Client.bind()*.

7.1.51 0.5.0 (2015-05-08)

- Handle attributes during parsing of the response values>
- Don't create empty soap objects when the root element is empty.
- Implement support for WSSE usernameToken profile including passwordText/passwordDigest.
- Improve XSD date/time related builtins.
- Various minor XSD handling fixes
- Use the correct soap-envelope XML namespace for the Soap 1.2 binding
- Use *application/soap+xml* as content-type in the Soap 1.2 binding
- **backwards incompatible**: Make cache part of the transport object instead of the client. This changes the call signature of the Client() class. (Marek Wywiał)
- Add the *verify* kwarg to the Transport object to disable ssl certificate verification. (Marek Wywiał)

7.1.52 0.4.0 (2016-04-17)

- Add defusedxml module for XML security issues
- Add support for choice elements
- Fix documentation example for complex types (Falk Schuetzenmeister)

7.1.53 0.3.0 (2016-04-10)

- Correctly handle recursion in WSDL and XSD files
- Add support for the XSD Any element
- Allow usage of shorthand prefixes when creating elements and types
- And more various improvements

7.1.54 0.2.5 (2016-04-05)

- Temporarily disable the HTTP binding support until it works properly
- Fix an issue with parsing SOAP responses with optional elements

7.1.55 0.2.4 (2016-04-03)

- Improve xsd.DateTime, xsd.Date and xsd.Time implementations by using the isodate module.
- Implement xsd.Duration

7.1.56 0.2.3 (2016-04-03)

- Fix xsd.DateTime, xsd.Date and xsd.Time implementations
- Handle NIL values correctly for simpletypes

7.1.57 0.2.2 (2016-04-03)

- Fix issue with initializing value objects (ListElements)
- Add new *zeep.helpers.serialize_object()* method
- Rename type attribute on value objects to *_xsd_type* to remove potential attribute conflicts

7.1.58 0.2.1 (2016-04-03)

- Support minOccurs 0 (optional elements)
- Automatically convert python datastructures to zeep objects for requests.
- Set default values for new zeep objects to None / [] (Element, ListElement)
- Add *Client.get_element()* to create custom objects

7.1.59 0.2.0 (2016-04-03)

- Proper support for XSD element and attribute forms (qualified/unqualified)
- Improved XSD handling
- Separate bindings for Soap 1.1 and Soap 1.2
- And again various other fixes

7.1.60 0.1.1 (2016-03-20)

- Various fixes to make the HttpBinding not throw errors during parsing
- More built-in xsd types
- Add support for *python -mzeep <wsdl>*
- Various other fixes

7.1.61 0.1.0 (2016-03-20)

Preview / Proof-of-concept release. Probably not suitable for production use :)

Python Module Index

Z

zeep, 29
zeep.helpers, 25
zeep.settings, 14
zeep.wsdl, 31
zeep.wsdl.attachments, 37
zeep.wsdl.bindings.http, 37
zeep.wsdl.bindings.soap, 38
zeep.wsdl.definitions, 34
zeep.wsdl.messages, 39
zeep.wsdl.messages.base, 40
zeep.wsdl.messages.http, 40
zeep.wsdl.messages.mime, 41
zeep.wsdl.messages.soap, 42
zeep.wsdl.parse, 35
zeep.wsdl.utils, 45
zeep.wsdl.wsdl, 31
zeep.xsd, 45
zeep.xsd.schema, 45
zeep.xsd.utils, 46
zeep.xsd.valueobjects, 46
zeep.xsd.visitor, 46

Index

A

AbstractMessage (*class in zeep.wsdl.definitions*), 34
AbstractOperation (*class in zeep.wsdl.definitions*), 34
AnyObject (*class in zeep*), 31
AnyObject (*class in zeep.xsd.valueobjects*), 46

B

bind() (*zeep.Client method*), 29
Binding (*class in zeep.wsdl.definitions*), 34

C

Client (*class in zeep*), 29
CompoundValue (*class in zeep.xsd.valueobjects*), 46
ConcreteMessage (*class in zeep.wsdl.messages.base*), 40
content (*zeep.wsdl.messages.base.SerializedMessage attribute*), 40
create_message() (*zeep.Client method*), 29
create_new_document() (*zeep.xsd.schema.Schema method*), 45
create_prefixed_name() (*in module zeep.xsd.utils*), 46
create_service() (*zeep.Client method*), 29
create_xml_soap_map() (*in module zeep.helpers*), 25

D

Definition (*class in zeep.wsdl.wsdl*), 32
deserialize() (*zeep.wsdl.messages.soap.DocumentMessage method*), 42
deserialize() (*zeep.wsdl.messages.soap.RpcMessage method*), 44
Document (*class in zeep.wsdl.wsdl*), 33
DocumentMessage (*class in zeep.wsdl.messages.soap*), 42

E

element (*zeep.wsdl.definitions.MessagePart attribute*), 34
elements (*zeep.xsd.schema.Schema attribute*), 45

G

get() (*zeep.Transport method*), 30
get_attribute() (*zeep.xsd.schema.Schema method*), 45
get_attribute() (*zeep.xsd.schema.SchemaDocument method*), 45
get_attribute_group() (*zeep.xsd.schema.Schema method*), 45
get_attribute_group() (*zeep.xsd.schema.SchemaDocument method*), 45
get_element() (*zeep.Client method*), 30
get_element() (*zeep.xsd.schema.Schema method*), 45
get_element() (*zeep.xsd.schema.SchemaDocument method*), 45
get_group() (*zeep.xsd.schema.Schema method*), 45
get_group() (*zeep.xsd.schema.SchemaDocument method*), 46
get_type() (*zeep.Client method*), 30
get_type() (*zeep.xsd.schema.Schema method*), 45
get_type() (*zeep.xsd.schema.SchemaDocument method*), 46
guess_xsd_type() (*in module zeep.helpers*), 25

H

headers (*zeep.wsdl.messages.base.SerializedMessage attribute*), 40
HttpBinding (*class in zeep.wsdl.bindings.http*), 37
HttpGetBinding (*class in zeep.wsdl.bindings.http*), 37
HttpOperation (*class in zeep.wsdl.bindings.http*), 37
HttpPostBinding (*class in zeep.wsdl.bindings.http*), 37

I

is_empty (*zeep.xsd.schema.Schema attribute*), 45

L

load() (*zeep.Transport method*), 31
load() (*zeep.xsd.schema.SchemaDocument method*), 46

M

match() (*zeep.wsdl.bindings.http.HttpGetBinding class method*), 37
match() (*zeep.wsdl.bindings.http.HttpPostBinding class method*), 37
match() (*zeep.wsdl.bindings.soap.SoapBinding class method*), 38
merge() (*zeep.xsd.schema.Schema method*), 45
MessagePart (*class in zeep.wsdl.definitions*), 34
MimeContent (*class in zeep.wsdl.messages.mime*), 41
MimeMultipart (*class in zeep.wsdl.messages.mime*), 42
MimeXML (*class in zeep.wsdl.messages.mime*), 41

N

Nil() (*in module zeep.helpers*), 25

O

Operation (*class in zeep.wsdl.definitions*), 34

P

parse() (*zeep.wsdl.bindings.http.HttpOperation class method*), 37
parse() (*zeep.wsdl.bindings.soap.SoapBinding class method*), 38
parse() (*zeep.wsdl.bindings.soap.SoapOperation class method*), 39
parse() (*zeep.wsdl.definitions.Operation class method*), 34
parse() (*zeep.wsdl.messages.soap.DocumentMessage class method*), 42
parse() (*zeep.wsdl.messages.soap.RpcMessage class method*), 44
parse_abstract_message() (*in module zeep.wsdl.parse*), 35
parse_abstract_operation() (*in module zeep.wsdl.parse*), 35
parse_binding() (*zeep.wsdl.wsdl.Definition method*), 32
parse_imports() (*zeep.wsdl.wsdl.Definition method*), 32
parse_messages() (*zeep.wsdl.wsdl.Definition method*), 32
parse_port() (*in module zeep.wsdl.parse*), 36
parse_port_type() (*in module zeep.wsdl.parse*), 36
parse_ports() (*zeep.wsdl.wsdl.Definition method*), 33
parse_service() (*in module zeep.wsdl.parse*), 36
parse_service() (*zeep.wsdl.wsdl.Definition method*), 33
parse_types() (*zeep.wsdl.wsdl.Definition method*), 33
path (*zeep.wsdl.messages.base.SerializedMessage attribute*), 40
Port (*class in zeep.wsdl.definitions*), 35
post() (*zeep.Transport method*), 31
post_xml() (*zeep.Transport method*), 31
process_reply() (*zeep.wsdl.bindings.soap.SoapBinding method*), 38

R

register_attribute() (*zeep.xsd.schema.SchemaDocument method*), 46
register_attribute_group() (*zeep.xsd.schema.SchemaDocument method*), 46
register_element() (*zeep.xsd.schema.SchemaDocument method*), 46
register_group() (*zeep.xsd.schema.SchemaDocument method*), 46
register_import() (*zeep.xsd.schema.SchemaDocument method*), 46
register_type() (*zeep.xsd.schema.SchemaDocument method*), 46
resolve() (*zeep.wsdl.messages.soap.DocumentMessage method*), 43
resolve() (*zeep.wsdl.messages.soap.RpcMessage method*), 44
resolve_imports() (*zeep.wsdl.wsdl.Definition method*), 33
RpcMessage (*class in zeep.wsdl.messages.soap*), 43

S

Schema (*class in zeep.xsd.schema*), 45
SchemaDocument (*class in zeep.xsd.schema*), 45
SchemaVisitor (*class in zeep.xsd.visitor*), 46
send() (*zeep.wsdl.bindings.http.HttpGetBinding method*), 37
send() (*zeep.wsdl.bindings.http.HttpPostBinding method*), 37
send() (*zeep.wsdl.bindings.soap.SoapBinding method*), 38
send_async() (*zeep.wsdl.bindings.soap.SoapBinding method*), 39
serialize() (*zeep.wsdl.messages.soap.DocumentMessage method*), 43
serialize() (*zeep.wsdl.messages.soap.RpcMessage method*), 45
serialize_object() (*in module zeep.helpers*), 25

SerializedMessage (*class in zeep.wsdl.messages.base*), 40
Service (*class in zeep.wsdl.definitions*), 35
service (*zeep.Client attribute*), 30
set_default_soapheaders() (*zeep.Client method*), 30
set_ns_prefix() (*zeep.Client method*), 30
Settings (*class in zeep.settings*), 14
settings() (*zeep.Transport method*), 31
Soap11Binding (*class in zeep.wsdl.bindings.soap*), 38
Soap12Binding (*class in zeep.wsdl.bindings.soap*), 38
SoapBinding (*class in zeep.wsdl.bindings.soap*), 38
SoapOperation (*class in zeep.wsdl.bindings.soap*), 39

T

Transport (*class in zeep*), 30
type (*zeep.wsdl.definitions.MessagePart attribute*), 34
type_factory() (*zeep.Client method*), 30
types (*zeep.xsd.schema.Schema attribute*), 45

U

UrlEncoded (*class in zeep.wsdl.messages.http*), 40
UrlReplacement (*class in zeep.wsdl.messages.http*), 40

V

visit_all() (*zeep.xsd.visitor.SchemaVisitor method*), 46
visit_annotation() (*zeep.xsd.visitor.SchemaVisitor method*), 47
visit_any() (*zeep.xsd.visitor.SchemaVisitor method*), 47
visit_any_attribute() (*zeep.xsd.visitor.SchemaVisitor method*), 47
visit_attribute() (*zeep.xsd.visitor.SchemaVisitor method*), 48
visit_attribute_group() (*zeep.xsd.visitor.SchemaVisitor method*), 48
visit_choice() (*zeep.xsd.visitor.SchemaVisitor method*), 48
visit_complex_content() (*zeep.xsd.visitor.SchemaVisitor method*), 49
visit_complex_type() (*zeep.xsd.visitor.SchemaVisitor method*), 49
visit_element() (*zeep.xsd.visitor.SchemaVisitor method*), 49
visit_extension_complex_content() (*zeep.xsd.visitor.SchemaVisitor method*), 50
visit_extension_simple_content() (*zeep.xsd.visitor.SchemaVisitor method*), 50
visit_group() (*zeep.xsd.visitor.SchemaVisitor method*), 50
visit_import() (*zeep.xsd.visitor.SchemaVisitor method*), 51
visit_include() (*zeep.xsd.visitor.SchemaVisitor method*), 51
visit_list() (*zeep.xsd.visitor.SchemaVisitor method*), 51
visit_notation() (*zeep.xsd.visitor.SchemaVisitor method*), 52
visit_restriction_complex_content() (*zeep.xsd.visitor.SchemaVisitor method*), 52
visit_restriction_simple_content() (*zeep.xsd.visitor.SchemaVisitor method*), 52
visit_restriction_simple_type() (*zeep.xsd.visitor.SchemaVisitor method*), 53
visit_schema() (*zeep.xsd.visitor.SchemaVisitor method*), 53
visit_sequence() (*zeep.xsd.visitor.SchemaVisitor method*), 53
visit_simple_content() (*zeep.xsd.visitor.SchemaVisitor method*), 54
visit_simple_type() (*zeep.xsd.visitor.SchemaVisitor method*), 54
visit_union() (*zeep.xsd.visitor.SchemaVisitor method*), 54
visit_unique() (*zeep.xsd.visitor.SchemaVisitor method*), 55

Z

zeep (*module*), 29
zeep.helpers (*module*), 25
zeep.settings (*module*), 14
zeep.wsdl (*module*), 31
zeep.wsdl.attachments (*module*), 37
zeep.wsdl.bindings.http (*module*), 37
zeep.wsdl.bindings.soap (*module*), 38
zeep.wsdl.definitions (*module*), 34
zeep.wsdl.messages (*module*), 39
zeep.wsdl.messages.base (*module*), 40
zeep.wsdl.messages.http (*module*), 40
zeep.wsdl.messages.mime (*module*), 41
zeep.wsdl.messages.soap (*module*), 42
zeep.wsdl.parse (*module*), 35
zeep.wsdl.utils (*module*), 45
zeep.wsdl.wsdl (*module*), 31
zeep.xsd (*module*), 45
zeep.xsd.schema (*module*), 45

zeep.xsd.utils (*module*), 46
zeep.xsd.valueobjects (*module*), 46
zeep.xsd.visitor (*module*), 46