Ethereum Name Record (ENR) library for Python Documentation

Release 0.5.0

The Ethereum Foundation

Nov 30, 2020

Contents

Python library for ENR (EIP-778) records

CHAPTER 1

Contents

1.1 Quickstart

1.1.1 ENR Creation

You can create an ENR record as follows.

```
>>> from eth_keys import keys
>>> from eth_enr import UnsignedENR, ENR
>>> private_key = keys.PrivateKey(b'unicornsrainbowsunicornsrainbows')
>>> unsigned_enr = UnsignedENR(
... sequence_number=1,
... kv_pairs={
...     b'id': b'v4',
...     b'secp256k1': private_key.public_key.to_compressed_bytes(),
...     b'unicorns': b'rainbows',
... })
>>> enr = unsigned_enr.to_signed_enr(private_key.to_bytes())
>>> enr
enr:-Ie4QNRDUVEiOYTwwki59qs5SY_ofKSCbFL2BuslZ9fsZXGEMOlfxkFGpojFUj_
↪ArnHMh4bv6E26frE1NII7z4xK9I0BgmlkgnY0iXNlY3AyNTZrMaEDvfDdonz3wUFd66sirz_
↪3a0oRlsc9rlKp0SQeHEkcC6iIdW5pY29ybnOIcmFpbmJvd3M
>>> enr == ENR.from_repr("enr:-Ie4QNRDUVEiOYTwwki59qs5SY_
↪ofKSCbFL2BuslZ9fsZXGEMOlfxkFGpojFUj_
↪ArnHMh4bv6E26frE1NII7z4xK9I0BgmlkgnY0iXNlY3AyNTZrMaEDvfDdonz3wUFd66sirz_
↪3a0oRlsc9rlKp0SQeHEkcC6iIdW5pY29ybnOIcmFpbmJvd3M")  # recover an ENR from it's text␣
↪representation
True
```

1.1.2 Storing ENR records

You can use the `eth_enr.ENRDB` to store ENR records. The underlying storage is flexible and accepts any dictionary-like object.

```
>>> from eth_keys import keys
>>> from eth_enr import UnsignedENR, ENRDB
>>> private_key = keys.PrivateKey(b'unicornsrainbowsunicornsrainbows')
>>> unsigned_enr = UnsignedENR(
... sequence_number=1,
... kv_pairs={
...     b'id': b'v4',
...     b'secp256k1': private_key.public_key.to_compressed_bytes(),
... })
>>> enr = unsigned_enr.to_signed_enr(private_key.to_bytes())
>>> enr_db = ENRDB({})
>>> enr_db.get_enr(enr.node_id)  # not yet in database
Traceback (most recent call last):
  File "/home/piper/.pyenv/versions/3.6.9/lib/python3.6/doctest.py", line 1330, in __
↪run
    compileflags, 1), test.globs)
  File "<doctest default[6]>", line 1, in <module>
    enr_db.get_enr(enr.node_id)  # not yet in database
  File "/home/piper/projects/eth-enr/eth_enr/enr_db.py", line 57, in get_enr
    return rlp.decode(self.db[self._get_enr_key(node_id)], sedes=ENR)  # type: ignore
KeyError: b'l?\x85b\xc8\x03\xbf\xae5\xa8\xf5K\x85\x82\xa2\x89V\xb9
↪%\x93M\x03\xdd\xb4Xu\xe1\x8e\x85\x93\x12\xc1:enr'
>>> enr_db.set_enr(enr)
>>> enr_db.get_enr(enr.node_id)
enr:-HW4QDBN_
↪uzB2BgXNgpjCN83hSE13oI46ZtFOmWnmYkGTZWrfRF6Yk60HcoiyuLDXqCTcj8fqk2DWetU2ZYJrXUEylIBgmlkgnY0iXNlY3Ay
↪3a0oRlsc9rlKp0SQeHEkcC6g
>>> updated_enr = UnsignedENR(
... sequence_number=2,
... kv_pairs={
...     b'id': b'v4',
...     b'secp256k1': private_key.public_key.to_compressed_bytes(),
... }).to_signed_enr(private_key.to_bytes())
>>> enr_db.set_enr(updated_enr)
>>> enr_db.set_enr(enr, raise_on_error=True)  # throws exception due to old sequence␣
↪number
Traceback (most recent call last):
  File "/home/piper/.pyenv/versions/3.6.9/lib/python3.6/doctest.py", line 1330, in __
↪run
    compileflags, 1), test.globs)
  File "<doctest default[11]>", line 1, in <module>
    enr_db.set_enr(enr)  # throws exception due to old sequence number
  File "/home/piper/projects/eth-enr/eth_enr/enr_db.py", line 51, in set_enr
    f"Cannot overwrite existing ENR ({existing_enr.sequence_number}) with old one "
eth_enr.exceptions.OldSequenceNumber: Cannot overwrite existing ENR (2) with old one␣
↪(1)
>>> assert enr_db.get_enr(updated_enr.node_id) == updated_enr
```

1.1.3 Using the ENRManager

The `eth_enr.ENRMAnager` automates creation, updating, and storage of ENR records.

```
>>> from eth_keys import keys
>>> from eth_enr import ENRManager, ENRDB
>>> private_key = keys.PrivateKey(b'unicornsrainbowsunicornsrainbows')
>>> manager = ENRManager(private_key, ENRDB({}))
```

(continues on next page)

(continued from previous page)

```
>>> manager.enr
enr:-HW4QDBN_
↪uzB2BgXNgpjCN83hSE13oI46ZtFOmWnmYkGTZWrfRF6Yk60HcoiyuLDXqCTcj8fqk2DWetU2ZYJrXUEylIBgmlkgnY0iXNlY3Ay
↪3a0oRlsc9rlKp0SQeHEkcC6g
>>> manager.enr.sequence_number
1
>>> manager.update((b'foo', b'bar'))
>>> manager.enr
enr:-H24QNUv1DBIpMITIUjJN8s7foWBJ33rR0liWCu4nVDaXk7ACcXpiMiFJHPC8UKTNkXfN3DXGwPX-
↪Q6KL1uMZwNeyGMCg2Zvb4NiYXKCaWSCdjSJc2VjcDI1NmsxoQO98N2ifPfBQV3rqyKvP_
↪drShGWxz2uUqnRJB4cSRwLqA
>>> manager.enr[b'foo']
b'bar'
>>> manager.enr.sequence_number
2
>>> manager.update((b'foo', None))  # `None` triggers removal of a key.
>>> manager.enr
enr:-HW4QFeb9Qg_RNSWamKytj4Eh2eICVKSauQfp4PMY45YQdGzAyFnLjZBU-IuktiGKGiEz2nbEo6w4qNOu_
↪D2Xdmr08gDgmlkgnY0iXNlY3AyNTZrMaEDvfDdonz3wUFd66sirz_3a0oRlsc9rlKp0SQeHEkcC6g
>>> manager.enr[b'foo']
Traceback (most recent call last):
  File "/home/piper/.pyenv/versions/3.6.9/lib/python3.6/doctest.py", line 1330, in __
↪run
    compileflags, 1), test.globs)
  File "<doctest default[10]>", line 1, in <module>
    manager.enr[b'foo']
  File "/home/piper/projects/eth-enr/eth_enr/enr.py", line 93, in __getitem__
    return self._kv_pairs[key]
KeyError: b'foo'
```

1.1.4 Querying ENR Records

You can use the `eth_enr.QueryableENRDB` which exposes the same API as `eth_enr.ENRDB` with one additional `eth_enr.QueryableENRDB.query()` method.

The `eth_enr.QueryableENRDB` operates on top of any SQLite3 database using the `sqlite3` standard library.

```
>>> import sqlite3
>>> from eth_keys import keys
>>> from eth_enr import UnsignedENR, QueryableENRDB
>>> from eth_enr.constraints import KeyExists
>>> private_key_a = keys.PrivateKey(b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')
>>> private_key_b = keys.PrivateKey(b'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB')
>>> private_key_c = keys.PrivateKey(b'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC')
>>> enr_a = UnsignedENR(
... sequence_number=1,
... kv_pairs={
...     b'id': b'v4',
...     b'secp256k1': private_key_a.public_key.to_compressed_bytes(),
...     b'unicorns': b'rainbows',
... }).to_signed_enr(private_key_a.to_bytes())
>>> enr_b = UnsignedENR(
... sequence_number=7,
... kv_pairs={
...     b'id': b'v4',
```

(continues on next page)

(continued from previous page)

```
...     b'secp256k1': private_key_b.public_key.to_compressed_bytes(),
...     b'unicorns': b'rainbows',
...     b'cupcakes': b'sparkles',
... }).to_signed_enr(private_key_b.to_bytes())
>>> enr_c = UnsignedENR(
... sequence_number=2,
... kv_pairs={
...     b'id': b'v4',
...     b'secp256k1': private_key_c.public_key.to_compressed_bytes(),
... }).to_signed_enr(private_key_c.to_bytes())
>>> connection = sqlite3.connect(":memory:")
>>> enr_db = QueryableENRDB(connection)
>>> enr_db.set_enr(enr_a)
>>> enr_db.set_enr(enr_b)
>>> enrs_with_unicorns = tuple(enr_db.query(KeyExists(b'unicorns')))
>>> assert enr_a in enrs_with_unicorns
>>> assert enr_b in enrs_with_unicorns
>>> assert enr_c not in enrs_with_unicorns
>>> enrs_with_cupcakes = tuple(enr_db.query(KeyExists(b'cupcakes')))
>>> assert enr_a not in enrs_with_cupcakes
>>> assert enr_b in enrs_with_cupcakes
>>> assert enr_c not in enrs_with_cupcakes
```

1.2 API

1.2.1 Abstract Base Classes

class eth_enr.abc.**CommonENRAPI**

Bases: collections.abc.Mapping, typing.Generic, abc.ABC

get_signing_message() → bytes

identity_scheme

node_id

public_key

sequence_number

class eth_enr.abc.**UnsignedENRAPI**

Bases: eth_enr.abc.CommonENRAPI

to_signed_enr(*private_key: bytes*) → eth_enr.abc.ENRAPI

class eth_enr.abc.**ENRAPI**

Bases: eth_enr.abc.CommonENRAPI

classmethod from_repr(*representation: str*, *identity_scheme_registry: collections.UserDict*) → eth_enr.abc.ENRAPI

signature

validate_signature() → None

class eth_enr.abc.**ENRManagerAPI**

Bases: abc.ABC

enr

update(**kv_pairs*) → None
Update the ENR record with the provided key/value pairs. Providing *None* for a value will result in the associated key being removed from the ENR.

class eth_enr.abc.**IdentitySchemeAPI**
Bases: abc.ABC

classmethod create_enr_signature(*enr: eth_enr.abc.CommonENRAPI*, *private_key: bytes*) → bytes
Create and return the signature for an ENR.

classmethod extract_node_id(*enr: eth_enr.abc.CommonENRAPI*) → NewType.<locals>.new_type
Retrieve the node id from an ENR.

classmethod extract_public_key(*enr: eth_enr.abc.CommonENRAPI*) → bytes
Retrieve the public key from an ENR.

classmethod validate_enr_signature(*enr: eth_enr.abc.ENRAPI*) → None
Validate the signature of an ENR.

classmethod validate_enr_structure(*enr: eth_enr.abc.CommonENRAPI*) → None
Validate that the data required by the identity scheme is present and valid in an ENR.

eth_enr.abc.**IdentitySchemeRegistryAPI**
alias of collections.UserDict

class eth_enr.abc.**ENRDatabaseAPI**
Bases: abc.ABC

delete_enr(*node_id: NewType.<locals>.new_type*) → None

get_enr(*node_id: NewType.<locals>.new_type*) → eth_enr.abc.ENRAPI

set_enr(*enr: eth_enr.abc.ENRAPI*, *raise_on_error: bool = False*) → None

class eth_enr.abc.**QueryableENRDatabaseAPI**
Bases: eth_enr.abc.ENRDatabaseAPI

query(**constraints*) → Iterable[eth_enr.abc.ENRAPI]

1.2.2 Classes

class eth_enr.enr.**ENR**(*sequence_number: int, kv_pairs: Mapping[bytes, Any], signature: bytes, identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*)
Bases: eth_enr.enr.ENRCommon, eth_enr.sedes.ENRSedes, eth_enr.abc.ENRAPI

classmethod from_repr(*representation: str, identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*) → eth_enr.enr.ENR

signature

validate_signature() → None

class eth_enr.enr.**UnsignedENR**(*sequence_number: int, kv_pairs: Mapping[bytes, Any], identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*)

Bases: eth_enr.enr.ENRCommon, eth_enr.abc.UnsignedENRAPI

to_signed_enr(*private_key: bytes*) → eth_enr.enr.ENR

class eth_enr.enr_manager.**ENRManager**(*private_key: eth_keys.datatypes.PrivateKey, enr_db: eth_enr.abc.ENRDatabaseAPI, kv_pairs: Optional[Mapping[bytes, bytes]] = None, identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*)

Bases: eth_enr.abc.ENRManagerAPI

enr

logger = <Logger eth_enr.ENRManager (WARNING)>

update(**kv_pairs*) → None

Update the ENR record with the provided key/value pairs. Providing *None* for a value will result in the associated key being removed from the ENR.

class eth_enr.identity_schemes.**IdentitySchemeRegistry**(***kwargs*)

Bases: collections.UserDict

register(*identity_scheme_class: Type[IdentitySchemeAPI]*) → Type[eth_enr.abc.IdentitySchemeAPI]

Class decorator to register identity schemes.

class eth_enr.identity_schemes.**V4IdentityScheme**

Bases: eth_enr.abc.IdentitySchemeAPI

classmethod create_enr_signature(*enr: eth_enr.abc.CommonENRAPI, private_key: bytes*) → bytes

Create and return the signature for an ENR.

classmethod extract_node_id(*enr: eth_enr.abc.CommonENRAPI*) → NewType.<locals>.new_type

Retrieve the node id from an ENR.

classmethod extract_public_key(*enr: eth_enr.abc.CommonENRAPI*) → bytes

Retrieve the public key from an ENR.

id = b'v4'

private_key_size = 32

public_key_enr_key = b'secp256k1'

classmethod validate_compressed_public_key(*public_key: bytes*) → None

classmethod validate_enr_signature(*enr: eth_enr.abc.ENRAPI*) → None

Validate the signature of an ENR.

classmethod validate_enr_structure(*enr: eth_enr.abc.CommonENRAPI*) → None

Validate that the data required by the identity scheme is present and valid in an ENR.

classmethod validate_signature(**, message_hash: bytes, signature: bytes, public_key: bytes*) → None

classmethod validate_uncompressed_public_key(*public_key: bytes*) → None

class eth_enr.identity_schemes.**V4CompatIdentityScheme**
Bases: eth_enr.identity_schemes.V4IdentityScheme

An identity scheme to be used for locally crafted ENRs representing remote nodes that don't support the ENR extension.

ENRs using this identity scheme have a zero-length signature.

classmethod create_enr_signature(*enr: eth_enr.abc.CommonENRAPI*, *private_key: bytes*) → bytes
Create and return the signature for an ENR.

id = b'v4-compat'

classmethod validate_enr_signature(*enr: eth_enr.abc.ENRAPI*) → None
Validate the signature of an ENR.

class eth_enr.enr_db.**ENRDB**(*db: MutableMapping[bytes, bytes]*, *identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*)
Bases: eth_enr.abc.ENRDatabaseAPI

delete_enr(*node_id: NewType.<locals>.new_type*) → None

get_enr(*node_id: NewType.<locals>.new_type*) → eth_enr.abc.ENRAPI

identity_scheme_registry

logger = <Logger eth_enr.ENRDB (WARNING)>

set_enr(*enr: eth_enr.abc.ENRAPI*, *raise_on_error: bool = False*) → None

class eth_enr.query_db.**QueryableENRDB**(*connection: sqlite3.Connection*, *identity_scheme_registry: collections.UserDict = {b'v4': <class 'eth_enr.identity_schemes.V4IdentityScheme'>, b'v4-compat': <class 'eth_enr.identity_schemes.V4CompatIdentityScheme'>}*)
Bases: eth_enr.abc.QueryableENRDatabaseAPI

An implementation of eth_enr.abc.QueryableENRDatabaseAPI on top of the sqlite3 module from the standard library.

For use with an in-memory database:

```
>>> connection = sqlite3.connect(":memory:")
>>> enr_db = QueryableENRDB(connection)
...
```

Or use with an on-disk database:

```
>>> connection = sqlite3.connect("/path/to/db.sqlite3")
>>> enr_db = QueryableENRDB(connection)
...
```

The database tables will lazily be created upon class instantiation if they are missing.

delete_enr(*node_id: NewType.<locals>.new_type*) → None
Delete ENR records with the given node_id

Raisees KeyError if there are no records with the given node_id

get_enr(*node_id: NewType.<locals>.new_type*) → eth_enr.abc.ENRAPI
Retrieve the ENR record with the highest sequence number for the given `node_id`

Raises `KeyError` if there are no records with the geven `node_id`

identity_scheme_registry

logger = <Logger eth_enr.ENRDB (WARNING)>

query(**constraints*) → Iterable[eth_enr.abc.ENRAPI]
Query the database for records that match the given constraints.

Support constraints:

- *KeyExists*
- *HasTCPIPv4Endpoint*
- *HasUDPIPv4Endpoint*
- *HasTCPIPv6Endpoint*
- *HasUDPIPv6Endpoint*

Return an iterator of matching ENR records. Only returns the record with the highest sequence number for each node_id.

set_enr(*enr: eth_enr.abc.ENRAPI*, *raise_on_error: bool = False*) → None
Write a record to the database.

Raise *eth_enr.exceptions.DuplicateRecord* if there is an different existing record with the same sequence number.

1.2.3 Constraints

class eth_enr.constraints.**KeyExists**(*key: bytes*)
Bases: `eth_enr.abc.ConstraintAPI`

Constrains ENR database queries to records which have a specified key.

```
>>> enr_db = ...
>>> from eth_enr.constraints import KeyExists
>>> for enr in enr_db.query(KeyExists(b"some-key")):
...     print("ENR: ", enr)
```

class eth_enr.constraints.**HasUDPIPv4Endpoint**
Bases: `eth_enr.abc.ConstraintAPI`

Constrains ENR database queries to records which have both the `"ip"` and `"udp"` keys.

```
>>> enr_db = ...
>>> from eth_enr.constraints import has_udp_ipv4_endpoint
>>> for enr in enr_db.query(has_udp_ipv4_endpoint):
...     print("ENR: ", enr)
```

class eth_enr.constraints.**HasUDPIPv6Endpoint**
Bases: `eth_enr.abc.ConstraintAPI`

Constrains ENR database queries to records which have both the `"ip6"` and `"udp6"` keys.

```
>>> enr_db = ...
>>> from eth_enr.constraints import has_udp_ipv6_endpoint
>>> for enr in enr_db.query(has_udp_ipv6_endpoint):
...     print("ENR: ", enr)
```

class eth_enr.constraints.**HasTCPIPv4Endpoint**

Bases: eth_enr.abc.ConstraintAPI

Constrains ENR database queries to records which have both the "ip" and "tcp" keys.

```
>>> enr_db = ...
>>> from eth_enr.constraints import has_tcp_ipv4_endpoint
>>> for enr in enr_db.query(has_tcp_ipv4_endpoint):
...     print("ENR: ", enr)
```

class eth_enr.constraints.**HasTCPIPv6Endpoint**

Bases: eth_enr.abc.ConstraintAPI

Constrains ENR database queries to records which have both the "ip6" and "tcp6" keys.

```
>>> enr_db = ...
>>> from eth_enr.constraints import has_tcp_ipv6_endpoint
>>> for enr in enr_db.query(has_tcp_ipv6_endpoint):
...     print("ENR: ", enr)
```

class eth_enr.constraints.**ClosestTo**(*node_id: NewType.<locals>.new_type*)

Bases: eth_enr.abc.ConstraintAPI

Constrains ENR database queries to return records proximate to a specific *node_id*

```
>>> enr_db = ...
>>> node_id = ...
>>> from eth_enr.constraints import ClosestTo
>>> for enr in enr_db.query(ClosestTo(node_id)):
...     print("ENR: ", enr)
```

1.2.4 Exceptions

class eth_enr.exceptions.**OldSequenceNumber**

Bases: eth_enr.exceptions.BaseENRException

Raised when trying to update an ENR record with a sequence number that is older than the latest sequence number we have seen

class eth_enr.exceptions.**DuplicateRecord**

Bases: eth_enr.exceptions.BaseENRException

Raised when trying to set an ENR record to a database that already has a different record with the same sequence number.

class eth_enr.exceptions.**UnknownIdentityScheme**

Bases: eth_enr.exceptions.BaseENRException

Raised when trying to instantiate an ENR with an unknown identity scheme

1.3 Release Notes

1.3.1 v0.1.0-alpha.1

- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER 2

Indices and tables

- genindex
- modindex

Index