InterlockLedgerAPI Documentation

Daniel Chino

Mar 29, 2022

CONTENTS:

This package is a python client to the InterlockLedger Node REST API v3.1.0. It connects to InterlockLedger nodes, allowing the creation of chains, interlocks, and storage of records and documents. This client requires the InterlockLedger API v7.2.0.

CHAPTER

ONE

THE INTERLOCKLEDGER

An InterlockLedger network is a peer-to-peer network of nodes. Each node runs the InterlockLedger software. All communication between nodes is point-to-point and digitally signed, but not mandatorily encrypted. This means that data is shared either publicly or on a need-to-know basis, depending on the application.

In the InterlockLedger, the ledger is composed of myriads of independently permissioned chains, comprised of blockchained records of data, under the control of their owners, but that are tied by Interlockings, that avoid them having their content/history being rewritten even by their owners. For each network the ledger is the sum of all chains in the participating nodes.

A chain is a sequential list of records, back chained with signatures/hashes to the previous records, so that no changes in them can go undetected. A record is tied to some enabled InterlockLedgerApplication (IL2App), that defines the metadata associate with it. The IL2App defines the constraints of a record as a public metadata, stored in the network genesis chain.

1.1 Setting Up the InterlockLedger API client

1.1.1 How to Use

To use the *il2_rest* package, you can add the il2_rest folder to your project and import the package.

```
>>> import il2_rest as il2
>>> node = il2.RestNode(cert_file='documenter.pfx', cert_pass='pwd')
```

1.1.2 Installing

The package can also be installed by running the following command on the `setup.py` folder:

```
$ pip3 install .
```

1.1.3 Dependencies

The *il2_rest* package was implemented using Python 3.6.9 and requires the following packages:

- colour (0.1.5)
- packaging (19.2)
- pyOpenSSL (19.1.0)
- requests (2.22.0)
- uri (2.0.1)
- pyilint (0.2.0)
- pyiltags>=0.0.1

1.2 Quickstart Tutorial

1.2.1 The Basics

To use the *il2_rest* client, you need to create an instance of the `RestNode` by passing a certificate file and the address of the node (default value is *localhost*).

Note: The certificate must be already imported to the InterlockLedger node and be permissioned on the desired chain. See the InterlockLedger node manual.

With the `RestNode` class, it is possible to retrieve details of the node, such as the list of valid apps in the network, peers, mirrors and chains.

```
>>> import il2_rest as il2
>>>
>>> node = il2.RestNode(cert_file='documenter.pfx', cert_pass='password', address=
↪'your.node.address', port=32020)
>>> print(node.details)
Node 'Node for il2tester on Apollo' Node!qh8D-FVQ8-2ng_EIDN8C9m3pOLAtz0BXKuCh9OBDr6U
Running il2 node#3.6.0 using [Message Envelope Wire Format #1] with Peer2Peer#2.1.0
Network Apollo
Color #20f9c7
Owner il2tester #Owner!yj...<REDACTED>...zk
Roles: Interlocking,Mirror,PeerRegistry,Relay,User
Chains: 20i...<REDACTED>..._fc, 5rA...<REDACTED>...Pso
```

To see and store records and documents, you need to use an instance of the `RestChain`. You can get `RestChain` instances by retrieving the list of chains in the network:

```
>>> for chain in node.chains:
...     print(chain)
...
Chain 'My first chain' #cA7CTUJxkcpGMpuGtg59kB9z5BllR-gQ4k4xBn8VAuo
Chain 'Second chain' #5rA_Fp9mhn3jb26G2Lsue5gWjxUdjLIWAs8Xvkg5Pso
Chain '3.6.2 chain name' #A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE
```

Or by its chain id:

```
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> print(chain)
Chain '3.6.2 chain name' #A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE
```

Besides retrieving and storing records and documents, the `RestChain` class also allows to manage the active apps in the chain, see/permit keys, and do interlocks.

1.2.2 Storing JSON Documents

The JSON Documents App allows you to store a custom JSON:

```
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> json_data = {
...     "field1" : 1,
...     "field2" : "Test",
...     "field3": [1,2,3],
...     "field4" : {
...         "value1" : 10,
...         "value2" : 20
...     }
... }
>>> new_json_document = chain.store_json_document(json_data)
>>> print(new_json_document)
```

1.2.3 Storing Multi-Documents

It is possible to store multiple documents in a single record of a chain. First you will need to begin a transaction:

```
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> resp = chain.documents_begin_transaction(comment ='Using parameters')
>>> transaction_id = resp.transactionId
```

Then, you can add as many files you wish using the transaction id:

```
>>> chain.documents_transaction_add_item(transaction_id, "item1.txt", "./test.txt",
↪"text/plain")
>>> chain.documents_transaction_add_item(transaction_id, "item2.txt", "./test2.txt",
↪"text/plain", "This file has a comment.")
```

When you are done, all you need to do is commit the transaction:

```
>>> locator = chain.documents_transaction_commit(transaction_id)
```

To download the files stored in a chain, you will need to use the locator of a multi-document record. You can store a single file of a multi-document record using the index of the file in the record:

```
>>> chain.download_single_document_at(locator, 0, '/path/to/download/')
```

Or you can download all files in a compressed in a single file:

```
>>> chain.download_documents_as_zip(locator, '/path/to/download/')
```

1.2.4 Creating Chains

If your are using a certificate with administration privileges, it is possible to create new chains. You can add a list of certificate to the chain's permissions by using the *apiCertificates* field with a list of `CertificatePermitModel`. The certificate (key) name must match (case insensitive) the name of the certificate imported in the IL2 node.

```
>>> node = RestNode(cert_file='admin.pfx', cert_pass='password', port=32020)
>>> certificate = PKCS12Certificate(
...     path='admin.pfx',
...     password='password'
... )
>>> permissions = [
...     AppPermissions(4),
...     AppPermissions(8)
... ]
>>> purposes = [
...     KeyPurpose.Action,
...     KeyPurpose.Protocol,
...     KeyPurpose.ForceInterlock
... ]
>>> cert_permit = CertificatePermitModel(
>>>     name='Certificate Name in IL2 Node',
>>>     permissions=permissions,
>>>     purposes=purposes,
>>>     pkcs12_certificate=certificate
>>> )
>>> new_chain = ChainCreationModel(
...     name='New chain name',
...     description='New chain',
...     additionalApps=[4,8],
...     managementKeyPassword='keyPassword',
...     emergencyClosingKeyPassword='closingPassword',
...     apiCertificates=[cert_permit]
... )
>>> resp = node.create_chain(new_chain)
>>> print(resp)
Chain 'New chain name' #cRPeHOITV_t1ZQS9CIL7Yi3djJ33ynZCdSRsEnOvX40
```

1.2.5 Managing Keys

You can see the list of keys permitted in the chain by using the following script:

```
>>> for key in chain.permitted_keys :
...     print(key)
...
Key 'emergency!A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE' Key!-
↪bLg6Skpj3Bhnn8A7VXkGnyED2oWHn9AhjpKiPL7sK0
    Purposes: [Protocol,Action]
    Actions permitted:
      App #0 Action 131
Key 'manager!A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE' Key!
↪QX5JpVthlQ5acCf3x05gCFyc5HEHQQwsbwnJDXyVROM
    Purposes: [Protocol,Action,KeyManagement]
    Actions permitted:
      App #2 Actions 500,501
      App #1 Actions 300,301
```

If you are using a certificate allowed to permit keys, you can permit other key in the chain:

Note: To permit other keys, the certificate must be already imported to the Interlockledger node with actions for App #2 and actions 500,501.

```
>>> from il2_rest.models import KeyPermitModel
>>> key_model = KeyPermitModel(app=4, appActions=[1000, 1001], key_id='Key!
↪MJ0kidltB324mfkiOG0aBlEocPA#SHA1',
...                 name='documenter', publicKey='PubKey!KPgQEPgItqh<...REDACTED...>
↪BZk4axWhFbTDrxADAQAB#RSA',
...                 purposes=[KeyPurpose.Action, KeyPurpose.Protocol])
>>> keys = chain.permit_keys([key_model])
>>> for key in keys :
...     print(keys)
...
Key 'emergency!A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE' Key!-
↪bLg6Skpj3Bhnn8A7VXkGnyED2oWHn9AhjpKiPL7sK0
    Purposes: [Protocol,Action]
    Actions permitted:
      App #0 Action 131
Key 'manager!A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE' Key!
↪QX5JpVthlQ5acCf3x05gCFyc5HEHQQwsbwnJDXyVROM
    Purposes: [Protocol,Action,KeyManagement]
    Actions permitted:
      App #2 Actions 500,501
      App #1 Actions 300,301
Key 'documenter' Key!MJ0kidltB324mfkiOG0aBlEocPA#SHA1
    Purposes: [Action,Protocol]
    Actions permitted:
      App #4 Actions 1000,1001
```

1.2.6 Permitting Apps

To check the active apps in the chain:

```
>>> print(chain.active_apps)
[0, 1, 2, 3, 5]
```

To permit new apps:

```
>>> apps = chain.permit_apps([4])
>>> print(apps)
[4]
```

1.2.7 Forcing Interlocks

The Interlocking is one of the concepts that grant immutability in IL2. They are made automatically by the network, this way there is no need for your application to worry about them. However, if you need to force an Interlocking, you can use the following code:

```
>>> from il2_rest.models import ForceInterlockModel
>>> force_model = ForceInterlockModel(targetChain=
↪'or7lzOGOvzH3GeNUTPqJI41CY0rVcEWgw6IEBmSSDxI')
>>> interlock_model = chain.force_interlock(model=force_model)
Interlocked chain or7lzOGOvzH3GeNUTPqJI41CY0rVcEWgw6IEBmSSDxI at record #11 (offset:␣
↪14308) with hash aneZJyR81OiqFzoQ0px4ZDFRCSNS9LzxbGUnueQKAtg#SHA256
```

If you need to check the interlockings of a chain:

```
>>> for interlock in chain.interlocks().items :
...     print(interlock)
```

1.3 The il2_rest package

This reference manual details the functions, modules and objects included in the *il2_rest* API.

1.3.1 Client module

This module has the classes needed to connect and communicate with the InterlockLedger REST API.

1.3.1.1 RestChain

class il2_rest.client.**RestChain**(*rest*, *chainId*, ***kwargs*)

Bases: object

REST API client to the InterlockLedger chain.

Note: It is not recomended to create an instance of *RestChain* outside of an instance of *RestNode*.

Parameters

- **rest** (*RestNode*) – Instance of the node.
- **chainId** (*il2_rest.models.ChainIdModel*) – Chain model.

id

Chain id.

Type str

name

Chain name.

Type str

licensingStatus

Licensing status.

Type str

property active_apps
Enumerate apps that are currently permitted on this chain.

Type list of int

add_record(*model*)
Add a new record.

Parameters **model** (*il2_rest.models.NewRecordModel*) – Model with the description of the new record.

Returns Added record information.

Return type *il2_rest.models.RecordModel*

Example

```
>>> node = RestNode(cert_file='recorder.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('cRPeHOITV_t1ZQS9CIL7Yi3djJ33ynZCdSRsEnOvX40')
>>> model = NewRecordModel(applicationId=1, payloadTagId=300,
...                payloadBytes=bytes([248, 52, 7, 5, 0, 0, 20, 2, 1, 4]))
>>> record = chain.add_record(model)
>>> print(record)
{
    "applicationId": 1,
    "chainId": "cRPeHOITV_t1ZQS9CIL7Yi3djJ33ynZCdSRsEnOvX40",
    "createdAt": "2020-02-13T18:59:50.9033962-03:00",
    "hash": "mAwaJCPH1c369GZLLXWsd_E7WkkZ2tdLS3LsZWBcPnw#SHA256",
    "payloadTagId": 300,
    "serial": 4,
    "type": "Data",
    "version": 2,
    "payloadBytes": "+DQHBQAAFAIBBA=="
}
```

add_record_as_json(*applicationId=None*, *payloadTagId=None*, *payload=None*, *rec_type=<RecordType.Data: 'Data'>*, *model=None*)
Add a new record with a payload encoded as JSON. The JSON value will be mapped to the payload tagged format as described by the metadata associated with the payloadTagId

Parameters

- **applicationId** (int) – Application id of the record.
- **payloadTagId** (int) – Payload tag id of the record.
- **payload** (int) – Payload data encoded as json
- **rec_type** (*il2_rest.enumerations.RecordType*) – Type of record.
- **model** (*il2_rest.models.NewRecordModelAsJson*) – Model with the description of the new record as JSON. **NOTE:** if model is not None, the other arguments will be ignored.

Returns Added record information.

Return type *il2_rest.models.RecordModel*

Example

```
>>> node = RestNode(cert_file='recorder.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('tdiy2HnWv-4a_h5T4Xy8l93CQ0lVkIeu2r5qgSlALMY')
>>> model = NewRecordModelAsJson(applicationId=1, payloadTagId=300, rec_json={
↪'tagId': 300,'version': 0, 'apps': [4]})
>>> record = chain.add_record_as_json(model=model)
>>> print(record)
{
    "applicationId": 1,
    "chainId": "tdiy2HnWv-4a_h5T4Xy8l93CQ0lVkIeu2r5qgSlALMY",
    "createdAt": "2020-02-13T18:56:44.3002447-03:00",
    "hash": "Y8Xb9FpTkgxj38xlwzcaZXm8fUq-NYxODVcyOQtzJ3c#SHA256",
    "payloadTagId": 300,
    "serial": 4,
    "type": "Data",
    "version": 2,
    "payload": {
        "tagId": 300,
        "version": 0,
        "apps": [
            4
        ]
    }
}
```

add_record_unpacked(*applicationId*, *payloadTagId*, *rec_bytes*, *rec_type=<RecordType.Data: 'Data'>*)

Add a new record with an unpacked payload. Payload inner bytes MUST go in the body, in binary form. These inner bytes will be prefixed with the payloadTagId and the lenght, both encoded as ILInt, as required to assemble the record effective payload.

Parameters

- **applicationId** (int) – Application id of the record.
- **payloadTagId** (int) – Payload tag id of the record.
- **rec_type** (*il2_rest.enumerations.RecordType*) – Type of record.
- **rec_bytes** (bytes) – Payload bytes.

Returns Added record information.

Return type *il2_rest.models.RecordModel*

Example

```
>>> node = RestNode(cert_file='recorder.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('VzCJczfgBeIiIBlnTRbmtsPriqwrkHqtF2yt8nhTcjM')
>>> record = chain.add_record_unpacked(applicationId=1, payloadTagId=300, rec_
↪bytes=bytes([5, 0, 0, 20, 2, 1, 4]))
>>> print(record)
{
    "applicationId": 1,
    "chainId": "VzCJczfgBeIiIBlnTRbmtsPriqwrkHqtF2yt8nhTcjM",
```

(continues on next page)

(continued from previous page)


```
    "createdAt": "2020-02-13T19:01:37.5175345-03:00",
    "hash": "cY7krS7BSJcBi7Ickq-u4iI6V6lYoKULfQtEZGJ-mC0#SHA256",
    "payloadTagId": 300,
    "serial": 4,
    "type": "Data",
    "version": 2,
    "payloadBytes": "+DQHBQAAFAIBBA=="
}
```


documents_begin_transaction(*comment=None*, *compression=None*, *generatePublicDirectory=None*, *iterations=None*, *encryption=None*, *password=None*, *model=None*)

Begin a transaction to store a set of documents. May rollback on timeout or errors.

Parameters

- **comment** (str) – Any additional information about the set of documents to be stored.
- **compression** (*il2_rest.enumerations.DocumentsCompression*) – Compression algorithm. The compression algorithm can be as follows:
  - NONE: No compression. Simply store the bytes;
  - GZIP: Compression of the data using the gzip standard;
  - BROTLI: Compression of the data using the brotli standard;
  - ZSTD: Compression of the data using the ZStandard from Facebook (In the future).
- **generatePublicDirectory** (bool) – If the publically viewable PublicDirectory field should be created.
- **iterations** (int) – Override for the number of PBE iterations to generate the key.
- **encryption** (str) – The encryption descriptor in the <pbe>-<hash>-<cipher>-<level> format Examples:
  - ”PBKDF2-SHA256-AES256-LOW”
  - ”PBKDF2-SHA512-AES256-MID”
  - ”PBKDF2-SHA256-AES128-HIGH”
- **password** (bytes) – Password as bytes if Encryption is not null.
- **model** (*il2_rest.models.DocumentsBeginTransactionModel*, optional) –

Returns Started transaction identifier and limits.

Return type il2_rest.models.DocumentsTransactionModel

Examples

Begin transaction using a *il2_rest.models.DocumentsBeginTransactionModel*:

```
>>> from il2_rest.models import DocumentsBeginTransactionModel
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> model = DocumentsBeginTransactionModel(chain='EbAfcWGwCwzuiEtSwIwYQYIHy-
↪g05CZl6jrcBAYuYRI',
...                                        comment='Using model')
>>> resp = chain.documents_transaction_metadata('EbAfcWGwCwzuiEtSwIwYQYIHy-
↪g05CZl6jrcBAYuYRIe')
>>> print(resp)
```

The same can be done passing all the information as parameters:

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> resp = chain.documents_begin_transaction(comment='Using parameters')
>>> print(resp)
```

documents_transaction_add_item(*transaction_id*, *name*, *filepath*, *content_type=None*, *comment=None*)

Adds another document to a pending transaction of multi-documents.

Parameters

- **transaction_id** (str) – Id of the ongoing transaction.
- **name** (str) – File name.
- **filepath** (str) – Path to the file to upload.
- **content_type** (str, optional) – File mime-type. If None, it will try to guess the mime-type based on the file extension.
- **comment** (str, optional) – Additional comment.

Returns True if success

Return type bool

Example

After beginning a transaction, you can add as many items as you wish: >>> node = RestNode(cert_file='documenter.pfx', cert_pass='password') >>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE') >>> resp = chain.documents_begin_transaction(comment='Using parameters') >>> transaction_id = resp.transactionId >>> chain.documents_transaction_add_item(transaction_id, "item1.txt", "./test.txt") >>> chain.documents_transaction_add_item(transaction_id, "item2.txt", "./test2.txt", comment="This file has a comment.")

documents_transaction_commit(*transaction_id*)

Store set of uploaded documents.

Note: Rementer to save the locator after commiting.

Parameters **transaction_id** (str) – Id of the ongoing transaction.

Returns Documents storage locator.

Return type str

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> resp = chain.documents_begin_transaction(comment='Using parameters')
>>> transaction_id = resp.transactionId
>>> chain.documents_transaction_add_item(transaction_id, "item1.txt", "text/
↪plain", "./test.txt"
>>> chain.documents_transaction_add_item(transaction_id, "item2.txt", "text/
↪plain", "./test2.txt", "This file has a comment."
>>> locator = chain.documents_transaction_commit(transaction_id)
```

documents_transaction_metadata(*locator*)

Retrieve the metadata for the set of documents from chain.

Parameters **locator** (str) – A Documents Storage Locator.

Returns Metadata associated to a Multi-Document Storage Locator

Return type *il2_rest.models.DocumentsMetadataModel*

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> resp = chain.documents_transaction_metadata('EbAfcWGwCwzuiEtSwIwYQYIHy-
↪g05CZl6jrcBAYuYRIe')
>>> print(resp)
```

documents_transaction_status(*transaction_id*)

Get the ongoing status of a transaction.

Parameters **transaction_id** (str) – Id of the transaction.

Returns Transaction identifier and limits.

Return type il2_rest.models.DocumentsTransactionModel

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> resp = chain.documents_transaction_status('IZqVW6p7z4hVdWzv')
>>> print(resp)
```

download_documents_as_zip(*locator*, *dst_path='./'*)

Download a compressed file with all documents to a folder (default: current folder).

Parameters

- **locator** (str) – A Documents Storage Locator.
- **dst_path** (str) – Download the file to this folder.

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> chain.download_documents_as_zip('EbAfcWGwCwzuiEtSwIwYQYIHy-
↪g05CZl6jrcBAYuYRIe', '/path/to/download/')
```

download_single_document_at (*locator*, *index*, *dst_path='./'*)

Download document by position from the set of documents to a folder (default: current folder).

Parameters

- **locator** (str) – A Documents Storage Locator.
- **index** (int) – Index of the file.
- **dst_path** (str) – Download the file to this folder.

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> chain.download_single_document_at('EbAfcWGwCwzuiEtSwIwYQYIHy-
↪g05CZl6jrcBAYuYRIe', 0, '/path/to/download/')
```

force_interlock (*model*)

Forces an interlock on a target chain.

Parameters **model** (*il2_rest.models.ForceInterlockModel*) – Force interlock command details.

Returns Interlocking details.

Return type *il2_rest.models.InterlockingRecordModel*

Example

```
>>> node = RestNode(cert_file='mykeymanager.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('VzCJczfgBeIiIBlnTRbmtsPriqwrkHqtF2yt8nhTcjM')
>>> model = ForceInterlockModel(targetChain='8fox30W54ZkzM-shfUeU5C7ad-_
↪fsf5nICwNpkCUk5w')
>>> interlocks = chain.force_interlock(model)
>>> for il in interlocks :
...     print(il)
...
Interlocked chain 8fox30W54ZkzM-shfUeU5C7ad-_fsf5nICwNpkCUk5w at record #14␣
↪(offset: 13671) with hash RyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo#SHA256
{
    "applicationId": 3,
    "chainId": "VzCJczfgBeIiIBlnTRbmtsPriqwrkHqtF2yt8nhTcjM",
    "createdAt": "2020-02-19T22:22:02.924546-03:46",
    "hash": "pGNSXOoI822Y_7F1ZNXw-xO02ufXXbrQjNXpTMkZJpQ#SHA256",
    "payloadTagId": 600,
    "serial": 7,
    "type": "Data",
    "version": 2,
```

(continues on next page)

(continued from previous page)

```
    "payloadBytes": "+QFgUgUBACsjAAEA8fox30W54ZkzM+shfUeU5C7ad+/
↪fsf5nICwNpkCUk5wKDgr5NG8nIgEARyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo=",
    "interlockedChainId": "8fox30W54ZkzM-shfUeU5C7ad-_fsf5nICwNpkCUk5w",
    "interlockedRecordHash": "RyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo
↪#SHA256",
    "interlockedRecordOffset": 13671,
    "interlockedRecordSerial": 14
}
```

interlocks (*howManyFromLast=0*, *page=0*, *pageSize=10*)
Get list of interlocks registered for the chain.

Parameters

- **howManyFromLast** (int) – How many interlocking records to return. If ommited or 0 returns all.
- **page** (int) – Page to return.
- **pageSize** (int) – Number of items per page. If 0 returns all.

Returns List of interlocks registered in the chain.

Return type *il2_rest.models.PageOfModel* of *il2_rest.models.InterlockingRecordModel*

json_document_at (*serial*)
Get a specific JSON document stored in the chain. :param serial: Serial number of the record. :type serial: int

Returns JSON document record.

Return type *il2_rest.models.JsonDocumentRecordModel*

permit_apps (*apps_to_permit*)
Add apps to the permitted list for the chain.

Parameters **apps_to_permit** (list of int) – List of apps (by number) to be permitted.

Returns Enumerate apps that are currently permitted on this chain.

Return type list of int

Example

```
>>> node = RestNode(cert_file='recorder.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> apps = chain.permit_apps([4])
>>> print(apps)
[4]
```

permit_keys (*keys_to_permit*)
Add keys to the permitted list for the chain.

Parameters **keys_to_permit** (list of *il2_rest.models.KeyPermitModel*) – List of keys to permitted.

Returns Enumerate keys that are currently permitted on chain.

Return type list of *il2_rest.models.KeyModel*

Example

```
>>> node = RestNode(cert_file='mykeymanager.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('20ic_KPTCIDfrlwQPKBHdKKp1a6ADaFtBvBjvFmf_fc')
>>> model_1 = KeyPermitModel(app=4, appActions=[1000, 1001], key_id='Key!
↪MJ0kidltB324mfkiOG0aBlEocPA#SHA1',
...                name='documenter', publicKey='PubKey!KPgQEPgItqh<...
↪REDACTED...>BZk4axWhFbTDrxADAQAB#RSA',
...                purposes=[KeyPurpose.Action, KeyPurpose.Protocol])
>>> model_2 = KeyPermitModel(key_id='Key!aWJWFHYDmUXCTCPIW2Ugih5l4XQ#SHA1',␣
↪name='recorder',
...                publicKey='PubKey!KPgQEPgItxD<...REDACTED...>
↪t1RvQCHPYtRADAQAB#RSA',
...                purposes=[KeyPurpose.Action, KeyPurpose.Protocol],
...                permissions=[AppPermissions(appId=1, actionIds=[300,301,306,
↪302,304,303,305,307])])
>>> keys = chain.permit_keys([model_1, model_2])
>>> for key in keys :
...     print(keys)
...
Key 'documenter' Key!MJ0kidltB324mfkiOG0aBlEocPA#SHA1
    Purposes: [Action,Protocol]
    Actions permitted:
      App #4 Actions 1000,1001
Key 'recorder' Key!aWJWFHYDmUXCTCPIW2Ugih5l4XQ#SHA1
    Purposes: [Action,Protocol]
    Actions permitted:
      App #1 Actions 300,301,306,302,304,303,305,307
Key 'mykeymanager' Key!-u07iGMWlkUm3WVBqS867AI-Lbw#SHA1
    Purposes: [KeyManagement,Action,Protocol]
    Actions permitted:
      App #2 Actions 500,501
Key 'emergency!20ic_KPTCIDfrlwQPKBHdKKp1a6ADaFtBvBjvFmf_fc' Key!
↪vckqYtMYIcetbunEJc4w-whbnqtZc9a9qlNp5PePm2E
    Purposes: [Protocol,Action]
    Actions permitted:
      App #0 Action 131
Key 'manager!20ic_KPTCIDfrlwQPKBHdKKp1a6ADaFtBvBjvFmf_fc' Key!hLZkEjBRofw1U-
↪JRkXfFdtBWfyM4sZNx8L3R5acakb4
    Purposes: [Protocol,Action,KeyManagement]
    Actions permitted:
      App #2 Actions 500,501
      App #1 Actions 300,301
```

property permitted_keys
Enumerate keys that are currently permitted on chain.

Type list of *il2_rest.models.KeyModel*

record_at (*serial*)
Get an specific record.

Parameters **serial** (int) – Record serial number.

Returns Record with the specific serial number.

Return type *il2_rest.models.RecordModel*

record_at_as_json (*serial*)

Get an specific record with payload mapped to json.

Parameters **serial** (int) – Record serial number.

Returns Record mapped to JSON with the specific serial number.

Return type *il2_rest.models.RecordModelAsJson*

records (*firstSerial=None*, *lastSerial=None*, *page=0*, *pageSize=10*, *lastToFirst=False*)
Get list of records starting from a given serial number.

Parameters

- **firstSerial** (int, optional) – Starting serial number.
- **lastSerial** (int, optional) – Last serial number.
- **page** (int, optional) – Page to return (Default is 0).
- **pageSize** (int, optional) – Number of items per page (Default is 10). If 0 returns all.
- **lastToFirst** (int, optional) – If True, return the list of records in reverse order (Default is False).

Returns List of records in the given interval.

Return type *il2_rest.models.PageOfModel* of *il2_rest.models.RecordModel*

records_as_json (*firstSerial=None*, *lastSerial=None*, *page=0*, *pageSize=10*, *lastToFirst=False*)
Get list of records with payload mapped to JSON starting from a given serial number.

Parameters

- **firstSerial** (int, optional) – Starting serial number.
- **lastSerial** (int, optional) – Last serial number.
- **page** (int, optional) – Page to return (Default is 0).
- **pageSize** (int, optional) – Number of items per page (Default is 10). If 0 returns all.
- **lastToFirst** (int, optional) – If True, return the list of records in reverse order (Default is False).

Returns List of records mapped to JSON in the given interval.

Return type *il2_rest.models.PageOfModel* of *il2_rest.models.RecordModelAsJson*

store_json_document (*payload*)
Store a JSON document record.

Parameters **payload** (dict) – A valid JSON.

Returns Added JSON document details.

Return type *il2_rest.models.JsonDocumentRecordModel*

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> json_data = {
...     "field1" : 1,
...     "field2" : "Test",
...     "field3": [1,2,3],
...     "field4" : {
...         "value1" : 10,
...         "value2" : 20
...     }
... }
>>> new_json_document = chain.store_json_document(json_data)
>>> print(new_json_document)
```

property summary
Chain details

Type *il2_rest.models.ChainSummaryModel*

1.3.1.2 RestNetwork

class il2_rest.client.**RestNetwork**(*rest*)
Bases: object

Informations about the node network.

Parameters **rest** (*RestNode*) – Node of the network.

property apps
List of valid apps in the network.

Type AppsModel

1.3.1.3 RestNode

class il2_rest.client.**RestNode**(*cert_file*, *cert_pass*, *port=32032*, *address='localhost'*, *verify_ca=True*)
Bases: object

REST API client to the InterlockLedger node.

You'll try to establish a bi-authenticated https connection with the configured node API address and port. The client-side certificate used to connect needs to be configured with the proper layered authorization role in the node configuration file and imported into a key permitted to update the chain that will be used.

Parameters

- **cert_file** (str) – Path to the .pfx certificate. Please refer to the InterlockLedger manual to see how to create and import the certificate into the node.
- **cert_pass** (str) – Password of the .pfx certificate.
- **port** (int) – Port number to connect.
- **address** (str) – Address of the node.
- **verify_ca** (bool) – If True, checks CA.

base_uri
The base URI address of the node.

Type uri.URI

network
Network information client.

Type *RestNetwork*

add_mirrors_of(*new_mirrors*)
Add new mirrors in this node.

Parameters **new_mirrors** (list of str) – List of chain ids you want to mirror.

Returns List of the chain information.

Return type list of *il2_rest.models.ChainIdModel*

property api_version
IL2 API version.

Type str

property certificate_name
Certificate friendly name.

Type str

chain_by_id(*chain_id*)
Get a chain by id.

Parameters **chain_id** (str) – Chain id.

Returns Chain instance with the corresponding id.

Return type *RestChain*

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password',␣
↪port=32020)
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> print(chain)
Chain '3.6.2 chain name' #A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE
```

property chains
List of chain instances.

Type list of *RestChain*

create_chain(*model*)
Create a new chain.

Parameters **model** (*il2_rest.models.ChainCreationModel*) – Model with the new chain attrbutes.

Returns Chain created model.

Return type *il2_rest.models.ChainCreatedModel*

Example

```
>>> node = RestNode(cert_file='admin.pfx', cert_pass='password', port=32020)
>>> certificate = PKCS12Certificate(
...     path='admin.pfx',
...     password='password'
... )
>>> permissions = [
...     AppPermissions(4),
...     AppPermissions(8)
... ]
>>> purposes = [
...     KeyPurpose.Action,
...     KeyPurpose.Protocol,
...     KeyPurpose.ForceInterlock
... ]
>>> cert_permit = CertificatePermitModel(
>>>     name='Certificate Name in IL2 Node',
>>>     permissions=permissions,
>>>     purposes=purposes,
>>>     pkcs12_certificate=certificate
>>> )
>>> new_chain = ChainCreationModel(
...     name='New chain name',
...     description='New chain',
...     additionalApps=[4,8],
...     managementKeyPassword='keyPassword',
...     emergencyClosingKeyPassword='closingPassword',
...     apiCertificates=[cert_permit]
... )
>>> resp = node.create_chain(new_chain)
>>> print(resp)
Chain 'New chain name' #cRPeHOITV_t1ZQS9CIL7Yi3djJ33ynZCdSRsEnOvX40
```

property details
Get node details.

Type *il2_rest.models.NodeDetailsModel*

property documents_config
Get documents upload configuration.

Type *il2_rest.models.DocumentUploadConfigurationModel*

interlocks_of(*chain*)
Get the list of interlocking records pointing to a target chain instance.

Parameters **chain** (str) – Chain id.

Returns List of interlockings.

Return type list of *il2_rest.models.InterlockingRecordModel*

Example

```
>>> node = RestNode(cert_file='documenter.pfx', cert_pass='password')
>>> interlocks = node.interlocks_of('8fox30W54ZkzM-shfUeU5C7ad-_
↪fsf5nICwNpkCUk5w')
>>> for interlock in interlocks :
...     print(interlock)
...
Interlocked chain 8fox30W54ZkzM-shfUeU5C7ad-_fsf5nICwNpkCUk5w at record #14␣
↪(offset: 13671) with hash RyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo#SHA256
{
    "applicationId": 3,
    "chainId": "A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE",
    "createdAt": "2020-02-26T23:17:03.018975-03:75",
    "hash": "0QjOJ-WQjauOF7qXeOxXabHxUgBR_KBNDZVDECbsszw#SHA256",
    "payloadTagId": 600,
    "serial": 9,
    "type": "Data",
    "version": 2,
    "payloadBytes": "+QFgUgUBACsjAAEA8fox30W54ZkzM+shfUeU5C7ad+/
↪fsf5nICwNpkCUk5wKDgr5NG8nIgEARyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo=",
    "interlockedChainId": "8fox30W54ZkzM-shfUeU5C7ad-_fsf5nICwNpkCUk5w",
    "interlockedRecordHash": "RyvOZIjnoUG4QX7FwQs3f6BqDfnOPb3txgXJNxLxtDo
↪#SHA256",
    "interlockedRecordOffset": 13671,
    "interlockedRecordSerial": 14
}
```

property mirrors
Get list of mirrors instances.

Type `list` of *RestChain*

property peers
Get list of known peers.

Type `list` of *il2_rest.models.PeerModel*

1.3.2 Models module

Resource models available in the InterlockLedger REST API.

1.3.2.1 CustomEncoder

class il2_rest.models.**CustomEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

Custom JSON encoder for the IL2 REST API models.

default(*obj*)
Set the behavior of the encoder depending on the type of obj.

1.3.2.2 BaseModel

class il2_rest.models.**BaseModel**
Bases: object

Base class for all models.

classmethod from_json(*json_data*)
Convert a dict (JSON like) to a *BaseModel* object.

Parameters **json_data** (dict) – JSON object to be converted.

Returns return an instance of the JSON model.

Return type *BaseModel*

json(*hide_null=True*, *return_as_str=False*)
Convert a BaseModel class to a dict (JSON like).

Parameters

- **hide_null** (bool, optional) – If True, discards every item (key, value) where value is None.
- **return_as_str** (bool, optional) – If True, return the JSON as a string instead of a dict.

Returns return obj as a JSON

Return type dict/str

classmethod to_json(*obj*, *hide_null=True*, *return_as_str=False*)
Convert an object to a dict (JSON like).

Parameters

- **obj** (list/dict/*BaseModel*) – Object to be converted to JSON.
- **hide_null** (bool, optional) – If True, discards every item (key, value) where value is None.
- **return_as_str** (bool, optional) – If True, return the JSON as a string instead of a dict.

Returns return obj as a JSON

Return type dict/str

1.3.2.3 AppsModel

class il2_rest.models.**AppsModel**(*network=None*, *validApps=[]*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

Details of the InterlockApps available in the chain.

Parameters

- **network** (str) – Network name.
- **validApps** (list of *PublishedApp*/list of dict) – List of currently valid apps for this network.
- ****kwargs** – Arbitrary keyword arguments.

network
Network name

Type str

validApps
Currently valid apps for this network

Type list of *PublishedApp*

class PublishedApp(*alternativeId=None*, *appVersion=None*, *description=None*, *app_id=None*, *name=None*, *publisherId=None*, *dataModels=None*, *publisherName=None*, *reservedILTagIds=None*, *simplifiedHashCode=None*, *start=None*, *version_=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

InterlockApp permitted in the chain.

alternativeId
Alternative id for the application.
Type int

appVersion
Application semantic version, with four numeric parts.
Type *version*

description
Description of the application.
Type str

id
Unique id for the application.
Type int

name
Application name.
Type str

publisherId
Publisher id, which is the identifier for the key the publisher uses to sign the workflow requests in its own chain. It should match the PublisherName
Type str

publisherName
Publisher name as registered in the Genesis chain of the network.
Type str

dataModels
The list of data models for the payloads of the records stored in the chains.
Type list of *DataModel*

reservedILTagIds
The list of ranges of ILTagIds to reserve for the application.
Type list of *il2_rest.util.LimitedRange*

simplifiedHashCode
Hash code.
Type int

start
The start date for the validity of the app, but if prior to the effective publication of the app will be

overridden with the publication date and time. If a string is passed, it will be automatically converted to datetime.datetime, as long as the string is in the 'yyyy-mm-ddTHH:MM:SS+HH:MM' format.
Type datetime.datetime/str

version
Version of the application.
Type int

alternativeId
Alternative id for the application.
Type int

appVersion
Application semantic version, with four numeric parts.
Type *version*

description
Description of the application.
Type str

id
Unique id for the application.
Type int

name
Application name.
Type str

publisherId
Publisher id, which is the identifier for the key the publisher uses to sign the workflow requests in its own chain. It should match the PublisherName
Type str

publisherName
Publisher name as registered in the Genesis chain of the network.
Type str

dataModels
The list of data models for the payloads of the records stored in the chains.
Type list of *DataModel*

reservedILTagIds
The list of ranges of ILTagIds to reserve for the application.
Type list of *il2_rest.util.LimitedRange*

simplifiedHashCode
Hash code.
Type int

start
The start date for the validity of the app, but if prior to the effective publication of the app will be overridden with the publication date and time.
Type datetime.datetime

version
Version of the application.
Type int

__eq__(*other*)
bool: Return True if self and other have the same id and appVersion.

__lt__(*other*)
bool: Return self.id < other.id. If self and other have the same id, return self.appVersion < other.appVersion.

__str__()
str: String representation of the published app.

property compositeName
Concatenation of the App's publisher name, name and version.
Type str

1.3.2.4 AppPermissions

class il2_rest.models.**AppPermissions**(*appId=None*, *actionIds=[]*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

App permissions

appId
App to be permitted (by number)

Type int

actionIds
App actions to be permitted by number.

Type list of int

__str__()
str: String representation of app permissions.

classmethod from_str(*permissions*)
Parse a string into an *AppPermissions* object.

Parameters **permissions** (str) – App permissions in the format used by the JSON response ('#<appId>,<actionId_1>,...,<actionId_n>').

Returns return an *AppPermissions* instance.

Return type *AppPermissions*

to_str()
str: String representation of app permissions in the JSON format ('#<appId>[,<actionId_1>,...,<actionId_n>]').

1.3.2.5 DataModel

class il2_rest.models.**DataModel**(*description=None*, *dataFields=None*, *indexes=None*, *payloadName=None*, *payloadTagId=None*, *version=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

Data model for the payloads and actions for the records the application stores in the chains.

description
Description of the data model.

Type str

dataFields
The list of data fields.

Type list of *DataModel.DataFieldModel*

indexes
List of indexes for records of this type.

Type list of *DataModel.DataIndexModel*

payloadName
Name of the record model.

Type str

payloadTagId
Tag id for this payload type. It must be a number in the reserved ranges.

Type int

version
Version of this data model, should start from 1.

Type int

class DataFieldModel(*cast=None*, *elementTagId=None*, *isOpaque=None*, *isOptional=None*, *description=None*, *Enumeration=None*, *enumerationAsFlags=None*, *name=None*, *serializationVersion=None*, *subDataFields=None*, *tagId=None*, *version=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

Metadata for field definition.

cast
Type of the data field.
Type *il2_rest.enumerations.DataFieldCast*

elementTagId
The type of the field in case it is an array.
Type int

isOpaque
If True the field is stored in raw bytes.
Type bool

isOptional
Indicate if data field is optional.
Type bool

name
Name of the data field.
Type str

serializationVersion
Data field definition version.
Type int

subDataFields
If the data field in composed of more fields, indicates the metadata of the subdata fields.
Type list of *DataModel.DataFieldModel*

tagId
Type of the field. (see tags in the InterlockLedger node documentation)
Type int

version
Version of the data field.
Type int

class DataIndexModel(*elements=None*, *isUnique=None*, *name=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Index of the data model.

elements
Elements of the index.
Type list of *DataModel.DataIndexModel.DataIndexElementModel*

isUnique
Indicate if the data field is unique.
Type bool

name
Name of the index.
Type str

class DataIndexElementModel(*descendingOrder=None*, *fieldPath=None*, *function=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Data index element.

descendingOrder
Indicate if the field is ordered in descending order.
Type bool

fieldPath
Path of the data field to be indexed.
Type str

function
To be defined.
Type str

1.3.2.6 ExportedKeyFile

class il2_rest.models.**ExportedKeyFile**(*keyFileBytes=None*, *keyFileName=None*, *keyName=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Key file info.

keyFileBytes
Key file in bytes.

Type bytes

keyFileName
Filename of the key.

Type str

keyName
Name of the key.

Type str

1.3.2.7 ChainIdModel

class il2_rest.models.**ChainIdModel**(*chain_id=None*, *name=None*, *licensingStatus=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Chain Id

id
Unique record id.

Type str

name
Chain name.

Type str

licensingStatus
Licensing status.

Type str

__eq__(*other*)
bool: Return self.id == other.id.

__hash__()
int: Hash representation of self.

__lt__(*other*)
bool: Return self.id < other.id.

__str__()
str: String representation of the *ChainIdModel*.

1.3.2.8 ChainCreatedModel

class il2_rest.models.**ChainCreatedModel**(*chain_id=None*, *name=None*, *keyFiles=[]*, ***kwargs*)

Bases: *il2_rest.models.ChainIdModel*

Chain created response.

id
Unique record id.

Type str

keyFiles
Emergency key file names.

Type list of *ExportedKeyFile*

name
Chain name.

Type str

1.3.2.9 ChainCreationModel

class il2_rest.models.**ChainCreationModel**(*name*, *emergencyClosingKeyPassword*, *managementKeyPassword*, *additionalApps=None*, *description=None*, *emergencyClosingKeyStrength=<KeyStrength.ExtraStrong: 'ExtraStrong'>*, *managementKeyStrength=<KeyStrength.Strong: 'Strong'>*, *keysAlgorithm=<Algorithms.RSA: 'RSA'>*, *operatingKeyStrength=<KeyStrength.Normal: 'Normal'>*, *parent=None*, *apiCertificates=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Chain creation parameters.

additionalApps
List of additional apps (only numeric ids).

Type list of int

description
Description (perhaps intended primary usage).

Type str

emergencyClosingKeyPassword
Emergency closing key password.

Type str

emergencyClosingKeyStrength
Emergency closing key strength of key.

Type *il2_rest.enumerations.KeyStrength*

managementKeyPassword
Key management key password.

Type str

managementKeyStrength
Key management strength of key.

Type *il2_rest.enumerations.KeyStrength*

keysAlgorithm
Keys algorithm.

Type *il2_rest.enumerations.Algorithms*

name
Name of the chain.

Type str

operatingKeyStrength
Operating key strength of key.

Type *il2_rest.enumerations.KeyStrength*

parent
Parent record Id.

Type str

apiCertificates (
obj:list of `CertificatePermitModel`): List of certificates to permit in the chain.

1.3.2.10 ChainSummaryModel

class il2_rest.models.**ChainSummaryModel**(*chain_id=None*, *name=None*, *activeApps=[]*, *description=None*, *isClosedForNewTransactions=False*, *lastRecord=None*, ***kwargs*)

Bases: *il2_rest.models.ChainIdModel*

Chain summary.

activeApps
List of active apps (only the numeric ids).

Type `list` of `int`

description
Description (perhaps intended primary usage).

Type `str`

isClosedForNewTransactions
Indicates if the chain accepts new records.

Type `bool`

lastRecord
Serial number of the last record.

Type `int`

1.3.2.11 DocumentUploadConfigurationModel

class il2_rest.models.**DocumentUploadConfigurationModel**(*defaultCompression=None*, *defaultEncryption=None*, *fileSizeLimit=None*, *iterations=None*, *permittedContentTypes=None*, *timeOutInMinutes=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Node configuration of uploaded documents.

Parameters

- **defaultCompression** (`str`) – Default compression algorithm.
- **defaultEncryption** (`str`) – Default encryption algorithm.
- **fileSizeLimit** (`int`) – Maximum file size.
- **iterations** (`int`) – Default number of PBE iterations to generate the key.
- **permittedContentTypes** (`list` of `str`) – List of content types mime-type/extension.
- **timeOutInMinutes** (`int`) – Timeout in minutes.

defaultCompression
Default compression algorithm.

Type str

defaultEncryption
Default encryption algorithm.

Type str

fileSizeLimit
Maximum file size.

Type int

iterations
Default number of PBE iterations to generate the key.

Type int

permittedContentTypes
List of content types mime-type/extension.

Type list of str

timeOutInMinutes
Timeout in minutes.

Type int

1.3.2.12 DocumentsBeginTransactionModel

class il2_rest.models.**DocumentsBeginTransactionModel**(*chain*, *comment=None*, *encryption=None*, *compression=None*, *generatePublicDirectory=None*, *iterations=None*, *password=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Parameters for starting a transaction to store many documents in a single InterlockLedger record.

Parameters

- **chain** (str) – Id of the chain where the set of documents should be stored.
- **comment** (str) – Any additional information about the set of documents to be stored.
- **compression** (*il2_rest.enumerations.DocumentsCompression*) – Compression algorithm.
- **encryption** (str) – The encryption descriptor in the <pbe>-<hash>-<cipher>-<level> format
- **generatePublicDirectory** (bool) – If the publically viewable PublicDirectory field should be created.
- **iterations** (int) – Override for the number of PBE iterations to generate the key.
- **password** (bytes) – Password as bytes if Encryption is not null.

chain
Id of the chain where the set of documents should be stored.

Type str

comment
Any additional information about the set of documents to be stored.

Type str

compression
Compression algorithm. The compression algorithm can be as follows:

- NONE: No compression. Simply store the bytes;
- GZIP: Compression of the data using the gzip standard;
- BROTLI: Compression of the data using the brotli standard;
- ZSTD: Compression of the data using the ZStandard from Facebook (In the future).

Type *il2_rest.enumerations.DocumentsCompression*

encryption
The encryption descriptor in the <pbe>-<hash>-<cipher>-<level> format. Examples:

- "PBKDF2-SHA256-AES256-LOW"
- "PBKDF2-SHA512-AES256-MID"
- "PBKDF2-SHA256-AES128-HIGH"

Type str

generatePublicDirectory
If the publically viewable PublicDirectory field should be created.

Type bool

iterations
Override for the number of PBE iterations to generate the key.

Type int

password
Password as bytes if Encryption is not null.

Type bytes

1.3.2.13 DocumentsMetadataModel

class il2_rest.models.**DocumentsMetadataModel**(*comment=None*, *compression=None*, *encryption=None*, *encryptionParameters=None*, *publicDirectory=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Metadata associated to a Multi-Document Storage Locator.

Parameters

- **comment** (str) – Any additional information about this set of documents
- **compression** (str) – Compression algorithm.
- **encryption** (str) – The encryption descriptor in the <pbe>-<hash>-<cipher>-<level> format.

- **encryptionParameters** (list of *EncryptionParameters*/list of str) – The parameters used to make the encryption of the set of documents.
- **publicDirectory** (*DirectoryEntry*/str) – List of stored documents.

comment
Any additional information about this set of documents

Type str

compression
Compression algorithm.

Type str

encryption
The encryption descriptor in the <pbe>-<hash>-<cipher>-<level> format.

Type str

encryptionParameters
The parameters used to make the encryption of the set of documents.

Type list of *EncryptionParameters*

publicDirectory
List of stored documents.

Type *DirectoryEntry*

class DirectoryEntry(*name=None*, *comment=None*, *mimeType=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

Entry for each stored document in this MultiDocument set

name
Document file name.
Type str

iterations
Any provided additional information about the document file.
Type str

mimeType
Mime Type for the document content
Type str

class EncryptionParameters(*iterations=None*, *salt=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

The parameters used to make the encryption of the set of documents.

iterations
Number of iterations to generate the key.
Type str

salt
Salt value to feed the cypher engine.
Type str

1.3.2.14 ForceInterlockModel

class il2_rest.models.**ForceInterlockModel**(*hashAlgorithm=<HashAlgorithms.SHA256: 'SHA256'>*, *minSerial=0*, *targetChain=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Force interlock command details.

hashAlgorithm
Hash algorithm to use.

Type *il2_rest.enumerations.HashAlgorithms*

minSerial
Required minimum of the serial of the last record in target chain whose hash will be pulled.

Type int

targetChain
Id of chain to be interlocked.

Type str

__str__()
(str): String representation of the interlock.

1.3.2.15 KeyModel

class il2_rest.models.**KeyModel**(*key_id=None*, *name=None*, *permissions=None*, *publicKey=None*, *purposes=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Key model

Parameters

- **key_id** (str) – Unique key id.
- **name** (str) – Key name.
- **permissions** (list of *AppPermissions*) – List of Apps and Corresponding Actions to be permitted by numbers.
- **publicKey** (str) – Key public key.
- **purposes** (list of *il2_rest.enumerations.KeyPurpose*/str) – Key valid purposes.
- ****kwargs** – Arbitrary keyword arguments.

id
Unique key id.

Type str

name
Key name.

Type str

permissions
List of Apps and Corresponding Actions to be permitted by numbers.

Type list of *AppPermissions*

publicKey
Key public key.

Type str

purposes
Key valid purposes.

Type list of *il2_rest.enumerations.KeyPurpose*/str

__str__()
(str): String representation of the key details.

property actionable
Return True if 'Action' is in the list of purposes.

Type (bool)

1.3.2.16 KeyPermitModel

class il2_rest.models.**KeyPermitModel**(*key_id=None*, *name=None*, *permissions=None*, *publicKey=None*, *purposes=[]*, *app=None*, *appActions=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Key to permit.

Parameters

- **key_id** (str) – Unique key id.
- **name** (str) – Key name.
- **permissions** (list of *AppPermissions*) – List of Apps and Corresponding Actions to be permitted by numbers.
- **publicKey** (str) – Key public key.
- **purposes** (list of *il2_rest.enumerations.KeyPurpose*/str) – Key valid purposes.
- **app** (int) – App to be permitted (by number). *Note*: If app and appActions is passed as parameter, permissions parameter will be ignored.
- **appActions** (list of int) – App actions to be permitted by number. *Note*: If app and appActions is passed as parameter, permissions parameter will be ignored.
- ****kwargs** – Arbitrary keyword arguments.

id
Unique key id.

Type str

name
Key name.

Type str

permissions
List of Apps and Corresponding Actions to be permitted by numbers.

Type list of *AppPermissions*

publicKey
Key public key.

Type str

purposes
Key valid purposes.

Type list of *il2_rest.enumerations.KeyPurpose*/str

1.3.2.17 NewRecordModelBase

class il2_rest.models.**NewRecordModelBase**(*applicationId=None*, *rec_type=<RecordType.Data: 'Data'>*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Base model for new Record.

applicationId
Application id this record is associated with.

Type int

rec_type
Block type. Most records are of the type 'Data'. Corresponds to the 'type' field in the JSON.

Type *il2_rest.enumerations.RecordType*

1.3.2.18 NewRecordModelAsJson

class il2_rest.models.**NewRecordModelAsJson**(*applicationId=None*, *rec_type=<RecordType.Data: 'Data'>*, *rec_json=None*, *payloadTagId=None*, ***kwargs*)

Bases: *il2_rest.models.NewRecordModelBase*

New record model to be added to the chain as a JSON.

JSON
The payload data matching the metadata for PayloadTagId.

Type dict

payloadTagId
The tag id for the payload, as registered for the application.

Type *il2_rest.enumerations.RecordType*

property to_query_string
Request query representation.

Type (str)

1.3.2.19 NewRecordModel

class il2_rest.models.**NewRecordModel**(*applicationId=None*, *rec_type=<RecordType.Data: 'Data'>*, *payloadBytes=None*, ***kwargs*)

Bases: *il2_rest.models.NewRecordModelBase*

New record model to be added to the chain as raw bytes.

payloadBytes
The payload in bytes. Must match the bytes schema of the application Id.

Type dict

1.3.2.20 NodeCommonModel

class il2_rest.models.**NodeCommonModel**(*color=None*, *node_id=None*, *name=None*, *network=None*, *ownerId=None*, *ownerName=None*, *roles=None*, *softwareVersions=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Node/Peer common details

color
Mapping color.

Type Color

id
Unique node id

Type str

name
Node name.

Type str

network
Network this node participates on.

Type str

ownerId
Node owner id

Type str

ownerName
Node owner name.

Type str

roles
List of active roles running in the node

Type list of str

softwareVersions
Version of software running the Node.

Type *Versions*

property fancy_color
Return the color as its name or the corresponding hexadecimal values.

Type (str)

1.3.2.21 NodeDetailsModel

class il2_rest.models.**NodeDetailsModel**(*color=None*, *node_id=None*, *name=None*, *network=None*, *ownerId=None*, *ownerName=None*, *roles=None*, *softwareVersions=None*, *chains=[]*, ***kwargs*)

Bases: *il2_rest.models.NodeCommonModel*

Node details

chains
List of owned records, only the ids

Type list of str

1.3.2.22 PeerModel

class il2_rest.models.**PeerModel**(*color=None*, *node_id=None*, *name=None*, *network=None*, *ownerId=None*, *ownerName=None*, *roles=None*, *softwareVersions=None*, *address=None*, *port=None*, *protocol=None*, ***kwargs*)

Bases: *il2_rest.models.NodeCommonModel*

Peer details.

address
Network address to contact the peer.

Type str

port
Port the peer is listening.

Type int

protocol
Network protocol the peer is listening.

Type *il2_rest.enumerations.NetworkProtocol*

1.3.2.23 RecordModelBase

class il2_rest.models.**RecordModelBase**(*applicationId=0*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=0*, *rec_type=0*, *version=None*, *reference=None*, *network=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Base model for records.

Parameters

- **applicationId** (int) – Application id this record is associated with.
- **chainId** (str) – Chain id that owns this record.

- **createdAt** (datetime.datetime/str) – Time of record creation. If a string is passed, it will be automatically converted to datetime.datetime, as long as the string is in the 'yyyy-mm-ddTHH:MM:SS+HH:MM' format.
- **rec_hash** (str) – Hash of the full encoded bytes of the record.
- **payloadTagId** (int) – The payload's TagId.
- **serial** (int) – Block serial number. For the first record this value is zero (0).
- **rec_type** (*il2_rest.enumerations.RecordType*) – Block type. Most records are of the type 'Data'. Corresponds to the 'type' field in the JSON.
- **version** (int) – Version of this record structure.
- **network** (str) – Network name this chain is part.
- **reference** (str) – Universal reference of this record.

applicationId
Application id this record is associated with.

Type int

chainId
Chain id that owns this record.

Type str

createdAt
Time of record creation.

Type datetime.datetime

hash
Hash of the full encoded bytes of the record.

Type str

payloadTagId
The payload's TagId.

Type int

serial
Block serial number. For the first record this value is zero (0).

Type int

type
Block type. Most records are of the type 'Data'. Corresponds to the 'type' field in the JSON.

Type *il2_rest.enumerations.RecordType*

version
Version of this record structure.

Type int

network
Network name this chain is part.

Type str

reference
Universal reference of this record.

Type str

__str__()
(str): JSON representation of the record as string.

1.3.2.24 RecordModel

class il2_rest.models.**RecordModel**(*applicationId=0*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=0*, *rec_type=0*, *version=None*, *reference=None*, *network=None*, *payloadBytes=None*, ***kwargs*)
Bases: *il2_rest.models.RecordModelBase*

Generic opaque record.

Parameters **payloadBytes** (bytes/str) – The payload's bytes. If loaded from JSON, can be input as a base64 string which will be decoded to bytes.

payloadBytes
The payload's bytes.

Type bytes

1.3.2.25 RecordModelAsJson

class il2_rest.models.**RecordModelAsJson**(*applicationId=0*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=0*, *rec_type=0*, *version=None*, *reference=None*, *network=None*, *payload=None*, ***kwargs*)
Bases: *il2_rest.models.RecordModelBase*

Record model as JSON.

payload
Payload bytes.

1.3.2.26 InterlockingRecordModel

class il2_rest.models.**InterlockingRecordModel**(*applicationId=0*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=0*, *rec_type=0*, *version=None*, *reference=None*, *network=None*, *payloadBytes=None*, *interlockedChainId=None*, *interlockedRecordHash=None*, *interlockedRecordOffset=None*, *interlockedRecordSerial=None*, ***kwargs*)
Bases: *il2_rest.models.RecordModel*

Interlocking details.

interlockedChainId
Interlocked Chain.

Type str

interlockedRecordHash
Interlock Record Hash.

Type str

interlockedRecordOffset
Interlocked Record Offset.

Type int

interlockedRecordSerial
Interlocked Record Serial.

Type int

__str__()
(str): String representation.

1.3.2.27 JsonDocumentRecordModel

class il2_rest.models.**JsonDocumentRecordModel**(*applicationId=0*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=0*, *rec_type=0*, *version=None*, *reference=None*, *network=None*, *jsonText=None*, *encryptedJson=None*, ***kwargs*)

Bases: *il2_rest.models.RecordModelBase*

Record to store JSON documents.

jsonText
JSON document as string.

Type str

encyptedJson
JSON Documents encrypted text.

Type *EncryptedTextModel*

1.3.2.28 EncryptedTextModel

class il2_rest.models.**EncryptedTextModel**(*cipher=None*, *cipherText=None*, *readingKeys=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

JSON Documents encrypted text.

cipher
Cipher algorithm used to cipher the text.

Type *il2_rest.enumerations.CipherAlgorithms*

cipherText
Encrypted text.

Type str

readingKeys
List of keys able to read this encrypted text.

Type list of *ReadiReadingKeyModel*

decode_with(*certificate*)
Decode the encrypted JSON Document text using the keys inside the certificate.

Parameters **certificate** (*il2_rest.util.PKCS12Certificate*) – PKCS12 certificate with the keys to decode the text.

Returns Decoded JSON.

Return type dict

Example

```
>>> node = RestNode(cert_file=cert_path,cert_pass=cert_pass, address=address,␣
↪port=port_number)
>>> chain = node.chains[0]
>>> json_body = {"attribute_1":"value_1", "number_1": 1}
>>> response = chain.store_json_document(json_body)
>>> pkcs12_cert = PKCS12Certificate(path=cert_path, password=cert_pass)
>>> response_json = response.encryptedJson.decode_with(pkcs12_cert)
>>> print(response_json)
{"attribute_1":"value_1", "number_1": 1}
```

1.3.2.29 ReadingKeyModel

class il2_rest.models.**ReadingKeyModel**(*encryptedIV=None*, *encryptedKey=None*, *publicKeyHash=None*, *readerId=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Keys able to read an encrypted JSON Document record.

encryptedIV
Encrypted AES256 IV.

Type str

encryptedKey
Encrypted AES256 key.

Type str

publicKeyHash
Public key hash in IL2 text representation.

Type str

readerId
Id of the key.

Type str

1.3.2.30 Versions

class il2_rest.models.**Versions**(*coreLibs=None*, *main=None*, *peer2peer=None*, *tags=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Versions for parts of the software.

coreLibs
Core libraries and il2apps version.

Type str

main
Interlockledger node daemon version.

Type str

peer2peer
Peer2Peer connectivity library version.

Type str

tags
Tag-Length-Value library version.

Type str

1.3.2.31 PageOfModel

class il2_rest.models.**PageOfModel**(*items=None*, *page=0*, *pageSize=None*, *totalNumberOfPages=None*, *itemClass=None*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

1.3.3 Enumerations module

Enumerations used in the InterlockLedger REST API.

1.3.3.1 Algorithms

class il2_rest.enumerations.**Algorithms**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the digital signature algorithms available in IL2.

DSA = 'DSA'

EcDSA = 'EcDSA'

EdDSA = 'EdDSA'

ElGamal = 'ElGamal'

RSA = 'RSA'

RSA15 = 'RSA15'

1.3.3.2 AutoName

class il2_rest.enumerations.**AutoName**(*value*)
Bases: enum.Enum

Base Enum class to automatically generate the enumerations values based on the enumeration name.

1.3.3.3 DataFieldCast

class il2_rest.enumerations.**DataFieldCast**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of casting options for DataField

DateTime = 'DateTime'

Integer = 'Integer'

NONE = 'None'

TimeSpan = 'TimeSpan'

1.3.3.4 CipherAlgorithms

class il2_rest.enumerations.**CipherAlgorithms**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the cipher algorithms available in IL2.

AES256 = 'AES256'

NONE = 'None'

1.3.3.5 HashAlgorithms

class il2_rest.enumerations.**HashAlgorithms**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the hash algorithms available in IL2.

Copy = 'Copy'

SHA1 = 'SHA1'

SHA256 = 'SHA256'

SHA3_256 = 'SHA3_256'

SHA3_512 = 'SHA3_512'

SHA512 = 'SHA512'

1.3.3.6 KeyPurpose

class il2_rest.enumerations.**KeyPurpose**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the purpose of keys in IL2.

Action = 'Action'

ChainOperation = 'ChainOperation'

ClaimSigner = 'ClaimSigner'

Encryption = 'Encryption'

ForceInterlock = 'ForceInterlock'

InvalidKey = 'InvalidKey'

KeyManagement = 'KeyManagement'

Protocol = 'Protocol'

1.3.3.7 KeyStrength

class il2_rest.enumerations.**KeyStrength**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the strength of keys.

Normal = 'Normal'
RSA 2048

Strong = 'Strong'
RSA 3072

ExtraStrong = 'ExtraStrong'
RSA 4096

MegaStrong = 'MegaStrong'
RSA 5120

SuperStrong = 'SuperStrong'
RSA 6144

HyperStrong = 'HyperStrong'
RSA 7172

UltraStrong = 'UltraStrong'
RSA 8192

1.3.3.8 NetworkProtocol

class il2_rest.enumerations.**NetworkProtocol**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the network protocols.

HTTPS_Proxied = 'HTTPS_Proxied'

Originator_Only = 'Originator_Only'

TCP_Direct = 'TCP_Direct'

TCP_Proxied = 'TCP_Proxied'

1.3.3.9 NetworkPredefinedPorts

class il2_rest.enumerations.**NetworkPredefinedPorts**(*value*)
Bases: enum.IntEnum

Enumeration of the default ports of the IL2 networks.

MainNet = 32032

MetaNet = 32036

TestNet_Apollo = 32020

TestNet_Janus = 32022

TestNet_Jupiter = 32030

TestNet_Liber = 32018

TestNet_Minerva = 32024

TestNet_Neptune = 32026

TestNet_Saturn = 32028

1.3.3.10 RecordType

class il2_rest.enumerations.**RecordType**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the types of Records available in IL2.

Closing = 'Closing'

Corrupted = 'Corrupted'

Data = 'Data'

EmergencyClosing = 'EmergencyClosing'

Root = 'Root'

1.3.3.11 DocumentsCompression

class il2_rest.enumerations.**DocumentsCompression**(*value*)
Bases: *il2_rest.enumerations.AutoName*

Enumeration of the compression algorithm.

BROTLI = 'BROTLI'

GZIP = 'GZIP'

NONE = 'NONE'

ZSTD = 'ZSTD'

1.3.4 Util module

Utility classes and functions for the InterlockLedger REST API.

1.3.4.1 LimitedRange

class il2_rest.util.**LimitedRange**(*start*, *count=1*, *end=None*)
Bases: object

A closed interval of integers represented by the notation '[start-end]'. If the range has only one value, the range is represented by '[start]'.

Parameters

- **start** (int) – Initial value of the interval
- **count** (int, optional) – How many elements are in the range
- **end** (int, optional) – If defined, define the end value of the interval

Raises **ValueError** – If 'count' is 0

start
Initial value of the interval

Type int

end
End value of the interval

Type int

__contains__(*item*)
Check if item is in self.

Parameters **item** (int/*LimitedRange*) – Item to check if is in self.

Returns Return item in self.

Return type bool

__eq__(*other*)
bool: Return self == other.

__hash__()
int: Hash representation of self.

__str__()
str: String representation of self.

property count
Number of elements in the interval.

Type int

overlaps_with(*other*)
Check if there is an overlap between the intervals of self and other.

Returns Return True if there is an overlap.

Return type bool

classmethod resolve(*text*)
Parses a string into a *LimitedRange*.

Parameters **text** (str) – String representing the range in the format of '[start]' or '[start-end]'.

Returns An instance of the LimitedRange represented by the *text*.

Return type *LimitedRange*

1.3.4.2 PKCS12Certificate

class il2_rest.util.**PKCS12Certificate**(*path*, *password*)
Bases: object

A PKCS12 certificate interface.

Parameters

- **path** (str) – Path to the .pfx certificate.
- **password** (str) – Password of the .pfx certificate.

property common_name
Certificate Common Name. If none found, return empty string.

Type str

decrypt(*cypher_text*)
Decode a encrypted message using RSA with SHA1.

Parameters **cypher_text** (bytes) – Encrypted message.

Returns Decrypted message.

Return type bytes

property friendly_name
Certificate friendly name (Not implemented).

Type str

has_pk()
Check if the certificate has a primary key.

Returns True if the certificate has a primary key.

Return type bool

property key_id
Id of the key.

Type str

property private_key
Certificate private key.

Type bytes

property pub_key_hash
Public key hash in IL2 text representation.

Type str

property public_certificate
Certificate public certificate.

Type bytes

property public_exponent
Public exponent.

Type int

property public_modulus
Public modulus.

Type int

1.3.4.3 null_condition_attribute

il2_rest.util.**null_condition_attribute**(*obj*, *attribute*)
Return the value of the item with key equals to attribute.

Parameters

- **obj** (dict) – Dictionary object.
- **attribute** (str) – Attribute name of obj.

Returns The value of the item. If obj is None, return None.

1.3.4.4 filter_none

il2_rest.util.**filter_none**(*d*)
Remove items of a dictionary with None values.

Parameters **d** (dict) – Dictionary object.

Returns Dictionary without None items.

Return type dict

1.3.4.5 string2datetime

il2_rest.util.**string2datetime**(*time_string*)
Convert a string to datetime object. The format of the string is as follows: 'yyyy-mm-ddTHH:MM:SS+HH:MM'.

Parameters **time_string** (str) – string with date and time.

Returns date time object.

Return type datetime.datetime

1.3.4.6 to_bytes

il2_rest.util.**to_bytes**(*value*)
Decodes value to bytes.

Parameters **value** – Value to decode to bytes

Returns

Return the value as bytes:

if type(value) is bytes, return value;

if type(value) is str, return the string encoded with UTF-8;

otherwise, returns bytes(value).

Return type bytes

1.3.4.7 build_query

il2_rest.util.**build_query**(*args_names*, *args_values*)
Transform a list of names and values in a HTTP query string.

Parameters

- **args_names** (list of str) – List of names.
- **args_values** (list) – List of values, must have same length of args_names.

Returns Query string.

Return type str

CHAPTER

TWO

ABOUT THIS DOCUMENTATION

This reference manual was partially created used using Sphinx and Google style docstrings. If you need/want to create this manual in another format (HTML, man, etc), you will need to install Sphinx and Sphinx-Napoleon extension:

```
$ pip3 install --user sphinx sphinxcontrib-napoleon2
```

To create an HTML version you can use the following instructions:

```
$ cd docs/
$ make html
```

To create the PDF version you can use the following instructions:

```
$ cd docs/
$ make latexpdf
```

Note: To create the PDF version, you must have a LaTeX builder (default is `pdflatex`) installed.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Z