InterlockLedgerAPI Documentation

Release

Daniel Chino

May 07, 2020

CONTENTS:

This package is a python client to the InterlockLedger Node REST API. It connects to InterlockLedger nodes, allowing the creation of chains, interlocks, and storage of records and documents. This client requires the InterlockLedger Node Server version 4.0.4.

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 Application, that defines the metadata associate with it, and the constraints defined in this public metadata, forcibly stored in the network genesis chain, is akin to validation that each correct implementation of the node software is able to enforce, but more importantly, any external logic can validate the multiple dimensions of validity for records/chains/interlockings/the ledger.

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)

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', 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 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 others 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
```

1.2.3 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]
```

1.2.4 Storing Documents

You can store documents using the *il2_rest*. There are three ways to store a document: plain text, bytes or file. To store a text document you can use the following script:

```
>>> doc_resp = chain.store_document_from_text(content = 'Plain text', name = 'text_
↪file.txt')
>>> print(doc_resp)
Document 'text_file.txt' [plain/text] uXKjPk_ftuMIFv90sJnjJJ0JYc5VoLjCIVaLPdhVP4c
↪#SHA256
```

If you need to store an array of bytes, you can use the following script:

```
>>> new_document = chain.store_document_from_bytes(doc_bytes = b'Bytes message!',␣
↪name = 'bytes_file.txt', content_type = 'plain/text')
>>> print(new_document)
Document 'bytes_file.txt' [plain/text] ZegBNUskzzJRqKvIuOiuhyhJvXJ5YxMJL99ONvqkcXs
↪#SHA256
```

It is also possible to store an array of bytes by using the `DocumentUploadModel`:

```
>>> from il2_rest.models import DocumentUploadModel
>>> model = DocumentUploadModel(name = 'other_bytes_file.txt', contentType = 'plain/
↪text')
>>> new_document = chain.store_document_from_bytes(doc_bytes = b'Other bytes message!
↪', model = model)
>>> print(new_document)
Document 'other_bytes_file.txt' [plain/text] wLQypXsHLV0H7RdNrrM3NvViA7W1-
↪9pcClPgWGMmF6Q#SHA256
```

Finally, you can store a file by passing its path:

```
>>> new_document = chain.store_document_from_file(file_path = './test.pdf', content_
↪type = 'application/pdf')
>>> print(new_document)
Document 'test.pdf' [application/pdf] tZpQvucMOi-FYHNQvI9UaOampVCUPtw3m0Z5TXwuF20
↪#SHA256
```

```
>>> from il2_rest.models import DocumentUploadModel
>>> model = DocumentUploadModel(name = 'my_test.txt', contentType = 'plain/text',␣
↪cipher = CipherAlgorithms.AES256)
>>> new_document = chain.store_document_from_file(file_path = './test.txt', model =␣
↪model)
>>> print(new_document)
Document 'my_test.txt' [plain/text] FukEkll0cTDSp4k4zJehM--5ZzjMz-LVeAsSeaMIeeg#SHA256
```

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.

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

id

str – Chain id.

name

str – Chain name.

active_apps

list of int – Enumerate apps that are currently permitted on this chain.

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

document_as_plain (*fileId*)
Retrieve document from chain as plain text.

Parameters **fileId** (str) – Unique id of the document file.

Returns Document content as a UTF-8 string.

Return type str

document_as_raw (*fileId*)
Retrieve document from chain as raw bytes.

Parameters **fileId** (str) – Unique id of the document file.

Returns Document model with content as raw bytes.

Return type *il2_rest.models.RawDocumentModel*

documents
list of *il2_rest.models.DocumentDetailsModel* – Enumerate documents that are stored on this chain.

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

interlocks
list of *il2_rest.models.InterlockingRecordModel* – List of interlocks registered in the chain.

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
```

permitted_keys
list of *il2_rest.models.KeyModel* – Enumerate keys that are currently permitted on chain.

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
list of *il2_rest.models.RecordModel* – List of records in the chain.

records_as_json
list of *il2_rest.models.RecordModelAsJson* – List of records in the chain with payload mapped to JSON.

records_from(*firstSerial*, *lastSerial=None*)
Get list of records starting from a given serial number.

Parameters

- **firstSerial** (int) – Starting serial number.
- **lastSerial** (int, optional) – Last serial number.

Returns List of records in the given interval.

Return type list of *il2_rest.models.RecordModel*

records_from_as_json(*firstSerial*, *lastSerial=None*)
Get list of records with payload mapped to JSON starting from a given serial number.

Parameters

- **firstSerial** (int) – Starting serial number.
- **lastSerial** (int, optional) – Last serial number.

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

Return type list of *il2_rest.models.RecordModelAsJson*

store_document_from_bytes(*doc_bytes*, *name=None*, *content_type=None*, *model=None*)
Store document on chain using bytes.

If more details is needed to upload the document, please use a *il2_rest.models.DocumentUploadModel* model.

Parameters

- **doc_bytes** (bytes) – Document bytes.
- **name** (str) – Document name (may be a file name with an extension).
- **content_type** (str) – Document content type (mime-type).
- **model** (*il2_rest.models.DocumentUploadModel*) – Model with the description of the new document. **NOTE:** if model is not None, the other arguments will be ignored.

Returns Added document details.

Return type *il2_rest.models.DocumentDetailsModel*

Examples

Adding a file document without specifying the name. The file name in the file_path will be used as the name of the document.

```
>>> node = RestNode(cert_file = 'documenter.pfx', cert_pass = 'password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> new_document = chain.store_document_from_bytes(doc_bytes = b'Bytes␣
↪message!', name = 'bytes_file.txt', content_type = 'plain/text')
>>> print(new_document)
Document 'bytes_file.txt' [plain/text]␣
↪ZegBNUskzzJRqKvIuOiuhyhJvXJ5YxMJL99ONvqkcXs#SHA256
```

Using the model to specify the description of the document.

```
>>> node = RestNode(cert_file = 'documenter.pfx', cert_pass = 'password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> model = DocumentUploadModel(name = 'other_bytes_file.txt', contentType =
↪'plain/text')
>>> new_document = chain.store_document_from_bytes(doc_bytes = b'Other bytes␣
↪message!', model = model)
>>> print(new_document)
Document 'other_bytes_file.txt' [plain/text] wLQypXsHLV0H7RdNrrM3NvViA7W1-
↪9pcClPgWGMmF6Q#SHA256
```

store_document_from_file (*file_path*, *content_type=None*, *name=None*, *model=None*)
Store document on chain using a file.

If more details is needed to upload the document, please use a *il2_rest.models.DocumentUploadModel* model.

Parameters

- **file_path** (bytes) – Filepath of the document file.
- **content_type** (str) – Document content type (mime-type).
- **name** (str, optional) – Document name (may be a file name with an extension). Can be derived from the file_path.
- **model** (*il2_rest.models.DocumentUploadModel*) – Model with the description of the new document. **NOTE:** if model is not None, the other arguments will be ignored.

Returns Added document details.

Return type *il2_rest.models.DocumentDetailsModel*

Examples

Adding a file document without specifying the name. The file name in the file_path will be used as the name of the document.

```
>>> node = RestNode(cert_file = 'documenter.pfx', cert_pass = 'password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> new_document = chain.store_document_from_file(file_path = './test.pdf',␣
↪content_type = 'application/pdf')
```

```
>>> print(new_document)
Document 'test.pdf' [application/pdf] tZpQvucMOi-
↪FYHNQvI9UaOampVCUPtw3m0Z5TXwuF20#SHA256
```

Using the model to specify the description of the document.

```
>>> node = RestNode(cert_file = 'documenter.pfx', cert_pass = 'password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> model = DocumentUploadModel(name = 'my_test.txt', contentType = 'plain/
↪text', cipher = CipherAlgorithms.AES256)
>>> new_document = chain.store_document_from_file(file_path = './test.txt',␣
↪model = model)
>>> print(new_document)
Document 'my_test.txt' [plain/text] FukEkl10cTDSp4k4zJehM--5ZzjMz-
↪LVeAsSeaMIeeg#SHA256
```

store_document_from_text (*content*, *name*, *content_type='plain/text'*)

Store document on chain using bytes.

If more details is needed to upload the document, please use a *il2_rest.models.DocumentUploadModel* model.

Parameters

- **doc_bytes** (bytes) – Document bytes.
- **content_type** (str) – Document content type (mime-type).
- **name** (str, optional) – Document name (may be a file name with an extension). Can be derived from the file_path.
- **model** (*il2_rest.models.DocumentUploadModel*) – Model with the description of the new document. **NOTE:** if model is not None, the other arguments will be ignored.

Returns Added document details.

Return type *il2_rest.models.DocumentDetailsModel*

Example

```
>>> node = RestNode(cert_file = 'documenter.pfx', cert_pass = 'password')
>>> chain = node.chain_by_id('A1wCG9hHhuVNb8hyOALHokYsWyTumHU0vRxtcK-iDKE')
>>> new_document = chain.store_document_from_text(content = 'Simple text',␣
↪name = 'document.txt')
>>> print(new_document)
Document 'document.txt' [plain/text] d_G2-zQ05L5QZ-
↪omHi7cfyJW1Ses4xovJuFoOUNnxNo#SHA256
```

summary

il2_rest.models.ChainSummaryModel – Chain details

RestNetwork

class il2_rest.client.**RestNetwork** (*rest*)

Bases: object

Informations about the node network.

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

apps
`AppsModel` – List of valid apps in the network.

RestNode

class `il2_rest.client.`**RestNode**(*cert_file*, *cert_pass*, *port=32032*, *address='localhost'*)
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.

base_uri
`uri.URI` – The base URI address of the node.

network
RestNetwork – Network information client.

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

Parameters **new_mirrors** (`list` of `str`) – List of mirrors chain ids.

Returns List of the chain information.

Return type `list` of *il2_rest.models.ChainIdModel*

certificate_name
`str` – Certificate friendly name.

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
```

chains
`list` of *RestChain* – List of chain instances.

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)
>>> new_chain = ChainCreationModel(name = 'New chain name', description =
↪'New chain',
...          managementKeyPassword = 'keyPassword',␣
↪emergencyClosingKeyPassword = 'closingPassword')
>>> resp = node.create_chain(new_chain)
>>> print(resp)
Chain 'New chain name' #cRPeHOITV_t1ZQS9CIL7Yi3djJ33ynZCdSRsEnOvX40
```

details
il2_rest.models.NodeDetailsModel – Get node details.

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
}
```

mirrors
list of *RestChain* – Get list of mirrors instances.

peers
list of *il2_rest.models.PeerModel* – Get list of known peers.

1.3.2 Models module

Resource models available in the InterlockLedger REST API.

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.

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`

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

`str` – Network name

validApps

`list` of *PublishedApp* – Currently valid apps for this network

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

`int` – Alternative id for the application.

appVersion

version – Application semantic version, with four numeric parts.

description

`str` – Description of the application.

id

`int` – Unique id for the application.

name

`str` – Application name.

publisherId
`str` – 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

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

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

reservedILTagIds
`list` of `il2_rest.util.LimitedRange` – The list of ranges of ILTagIds to reserve for the application.

simplifiedHashCode
`int` – 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.

start
`datetime.datetime` – 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.

version
`int` – Version of the application.

__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.

compositeName
`str` – Concatenation of the App's publisher name, name and version.

AppPermissions

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

App permissions

appId
`int` – App to be permitted (by number)

actionIds
`list` of `int` – App actions to be permitted by number.

__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>').

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

`str` – Description of the data model.

dataFields

`list` of *DataModel.DataFieldModel* – The list of data fields.

indexes

`list` of *DataModel.DataIndexModel* – List of indexes for records of this type.

payloadName

`str` – Name of the record model.

payloadTagId

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

version

`int` – Version of this data model, should start from 1.

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

il2_rest.enumerations.DataFieldCast – Type of the data field.

elementTagId

`int` – The type of the field in case it is an array.

isOpaque

`bool` – If `True` the field is stored in raw bytes.

isOptional

`bool` – Indicate if data field is optional.

name

`str` – Name of the data field.

serializationVersion

`int` – Data field definition version.

subDataFields

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

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

version
`int` – Version of the data field.

class DataIndexModel(*elements=None*, *isUnique=None*, *name=None*, ***kwargs*)
Bases: `il2_rest.models.BaseModel`

Index of the data model.

elements
`list` of `DataModel.DataIndexModel.DataIndexElementModel` – Elements of the index.

isUnique
`bool` – Indicate if the data field is unique.

name
`str` – Name of the index.

class DataIndexElementModel(*descendingOrder=None*, *fieldPath=None*, *function=None*, ***kwargs*)
Bases: `il2_rest.models.BaseModel`

Data index element.

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

fieldPath
`str` – Path of the data field to be indexed.

function
`str` – To be defined.

ExportedKeyFile

class il2_rest.models.**ExportedKeyFile**(*keyFileBytes=None*, *keyFileName=None*, *keyName=None*, ***kwargs*)
Bases: `il2_rest.models.BaseModel`

Key file info.

keyFileBytes
`bytes` – Key file in bytes.

keyFileName
`str` – Filename of the key.

keyName
`str` – Name of the key.

ChainIdModel

class il2_rest.models.**ChainIdModel**(*chain_id=None*, *name=None*, ***kwargs*)
Bases: `il2_rest.models.BaseModel`

Chain Id

id
`str` – Unique record id

name
`str` – Chain name

__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*.

ChainCreatedModel

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

Chain created response.

id
`str` – Unique record id.

keyFiles
`list` of *ExportedKeyFile* – Emergency key file names.

name
`str` – Chain name.

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*, ***kwargs*)
Bases: *il2_rest.models.BaseModel*

Chain creation parameters.

additionalApps
`list` of `int` – List of additional apps (only numeric ids).

description
`str` – Description (perhaps intended primary usage).

emergencyClosingKeyPassword
`str` – Emergency closing key password.

emergencyClosingKeyStrength
: `il2_rest.enumerations.KeyStrength` – Emergency closing key strength of key.

managementKeyPassword
: `str` – Key management key password.

managementKeyStrength
: `il2_rest.enumerations.KeyStrength` – Key management strength of key.

keysAlgorithm
: `il2_rest.enumerations.Algorithms` – Keys algorithm.

name
: `str` – Name of the chain.

operatingKeyStrength
: `il2_rest.enumerations.KeyStrength` – Operating key strength of key.

parent
: `str` – Parent record Id.

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 `int` – List of active apps (only the numeric ids).

description
: `str` – Description (perhaps intended primary usage).

isClosedForNewTransactions
: `bool` – Indicates if the chain accepts new records.

lastRecord
: `int` – Serial number of the last record.

DocumentBaseModel

class il2_rest.models.**DocumentBaseModel**(*cipher=<CipherAlgorithms.NONE: 'None'>*, *keyId=None*, *name=None*, *previousVersion=None*, ***kwargs*)

Bases: `il2_rest.models.BaseModel`

Document base model.

cipher
: `il2_rest.enumerations.CipherAlgorithms` – Cipher algorithm used to cipher the document.

keyId
: `str` – Unique id of key that ciphers this document.

name
: `str` – Document name, may be a file name with an extension.

previousVersion
`str` – A reference to a previous version of this document (ChainId and RecordNumber).

is_ciphered
(`bool`) – Return True if the document is ciphered.

DocumentDetailsModel

class il2_rest.models.**DocumentDetailsModel**(*cipher=<CipherAlgorithms.NONE: 'None'>, keyId=None, name=None, previousVersion=None, contentType=None, fileId=None, physicalDocumentID=None, **kwargs*)

Bases: *il2_rest.models.DocumentBaseModel*

Document details.

contentType
`str` – Document content type (mime-type).

fileId
`str` – Unique id of the document derived from its content. The same content stored in different chains will have the same FileId.

physicalDocumentID
`str` – Compound id for this document as stored in this chain.

__str__()
(`str`): String representation of the document: 'Document '{name}' [{contentType}] {fileId}'.

is_plain_text
(`bool`) – Return True if the content type is plain/text.

DocumentUploadModel

class il2_rest.models.**DocumentUploadModel**(*cipher=<CipherAlgorithms.NONE: 'None'>, keyId=None, name=None, previousVersion=None, contentType=None, **kwargs*)

Bases: *il2_rest.models.DocumentBaseModel*

Document model used to upload/post documents in the chain.

contentType
`str` – Document content type (mime-type).

to_query_string
(`str`) – Request query representation.

RawDocumentModel

class il2_rest.models.**RawDocumentModel**(*contentType=None, content=None, name=None, **kwargs*)

Bases: *il2_rest.models.BaseModel*

Document as raw data.

Parameters

- **contentType** (`str`) – Document content type (mime-type).

- **content** (bytes/bytes) – Content of the document in raw bytes. If loaded from JSON, can be input as a base64 string which will be decoded to bytes.
- **name** (str) – Document name, may be a file name with an extension.

contentType
str – Document content type (mime-type).

content
bytes – Content of the document in raw bytes.

name
str – Document name, may be a file name with an extension.

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
il2_rest.enumerations.HashAlgorithms – Hash algorithm to use.

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

targetChain
str – Id of chain to be interlocked.

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

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
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.

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

actionable
(`bool`) – Return True if 'Action' is in the list of purposes.

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
`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.

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
`int` – Application id this record is associated with.

rec_type
il2_rest.enumerations.RecordType – Block type. Most records are of the type 'Data'. Corresponds to the 'type' field in the JSON.

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
`dict` – The payload data matching the metadata for PayloadTagId.

payloadTagId
il2_rest.enumerations.RecordType – The tag id for the payload, as registered for the application.

to_query_string
(`str`) – Request query representation.

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
`dict` – The payload in bytes. Must match the bytes schema of the application Id.

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
`Color` – Mapping color.

id
`str` – Unique node id

name
`str` – Node name.

network
`str` – Network this node participates on.

ownerId
`str` – Node owner id

ownerName
`str` – Node owner name.

roles
`list` of `str` – List of active roles running in the node

softwareVersions
`Versions` – Version of software running the Node.

fancy_color
(`str`) – Return the color as its name or the corresponding hexadecimal values.

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 `str` – List of owned records, only the ids

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
`str` – Network address to contact the peer.

port
`int` – Port the peer is listening.

protocol
`il2_rest.enumerations.NetworkProtocol` – Network protocol the peer is listening.

RecordModelBase

class il2_rest.models.**RecordModelBase**(*applicationId=None*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=None*, *rec_type=None*, *version=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) – Time of record creation.
- **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.

applicationId
int – Application id this record is associated with.

chainId
str – Chain id that owns this record.

createdAt
datetime.datetime – Time of record creation.

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).

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.

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

RecordModel

class `il2_rest.models.`**`RecordModel`**(*applicationId=None*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=None*, *rec_type=None*, *version=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`
`bytes` – The payload's bytes.

RecordModelAsJson

class `il2_rest.models.`**`RecordModelAsJson`**(*applicationId=None*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=None*, *rec_type=None*, *version=None*, *payload=None*, ***kwargs*)

Bases: *`il2_rest.models.RecordModelBase`*

Record model as JSON.

`payload`
Payload bytes.

InterlockingRecordModel

class `il2_rest.models.`**`InterlockingRecordModel`**(*applicationId=None*, *chainId=None*, *createdAt=None*, *rec_hash=None*, *payloadTagId=None*, *serial=None*, *rec_type=None*, *version=None*, *payloadBytes=None*, *interlockedChainId=None*, *interlockedRecordHash=None*, *interlockedRecordOffset=None*, *interlockedRecordSerial=None*, ***kwargs*)

Bases: *`il2_rest.models.RecordModel`*

Interlocking details.

`interlockedChainId`
`str` – Interlocked Chain.

`interlockedRecordHash`
`str` – Interlock Record Hash.

`interlockedRecordOffset`
`int` – Interlocked Record Offset.

`interlockedRecordSerial`
`int` – Interlocked Record Serial.

`__str__`()
(`str`): String representation.

Versions

class il2_rest.models.**Versions**(*coreLibs=None*, *messageEnvelopeWireFormat=None*, *node=None*, *peer2peer=None*, ***kwargs*)

Bases: *il2_rest.models.BaseModel*

Versions for parts of the software.

coreLibs
`str` – Core libraries and il2apps version.

messageEnvelopeWireFormat
`str` – Message envelope wire format version.

node
`str` – Interlockledger node daemon version.

peer2peer
`str` – Peer2Peer connectivity library version.

1.3.3 Enumerations module

Enumerations used in the InterlockLedger REST API.

Algorithms

class il2_rest.enumerations.**Algorithms**

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'

AutoName

class il2_rest.enumerations.**AutoName**

Bases: `enum.Enum`

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

DataFieldCast

class il2_rest.enumerations.**DataFieldCast**

Bases: *il2_rest.enumerations.AutoName*

Enumeration of casting options for DataField

DateTime = 'DateTime'

Integer = 'Integer'

NONE = 'None'

TimeSpan = 'TimeSpan'

CipherAlgorithms

class il2_rest.enumerations.**CipherAlgorithms**

Bases: *il2_rest.enumerations.AutoName*

Enumeration of the cipher algorithms available in IL2.

AES256 = 'AES256'

NONE = 'None'

HashAlgorithms

class il2_rest.enumerations.**HashAlgorithms**

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'

KeyPurpose

class il2_rest.enumerations.**KeyPurpose**

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'

KeyStrength

class il2_rest.enumerations.**KeyStrength**

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

NetworkProtocol

class il2_rest.enumerations.**NetworkProtocol**

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'

NetworkPredefinedPorts

class il2_rest.enumerations.**NetworkPredefinedPorts**

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

RecordType

class il2_rest.enumerations.**RecordType**

Bases: *il2_rest.enumerations.AutoName*

Enumeration of the types of Records available in IL2.

Closing = 'Closing'

Corrupted = 'Corrupted'

Data = 'Data'

EmergencyClosing = 'EmergencyClosing'

Root = 'Root'

1.3.4 Util module

Utility classes and functions for the InterlockLedger REST API.

LimitedRange

class il2_rest.models.**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

`int` – Initial value of the interval

end

`int` – End value of the interval

__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.

`count`
`int` – Number of elements in the interval.

`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`*

null_condition_attribute

`il2_rest.models.`**`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.

filter_none

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

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

Returns Dictionary without None items.

Return type `dict`

string2datetime

`il2_rest.models.`**`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`

to_bytes

`il2_rest.models.`**`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`

CHAPTER

TWO

ABOUT THIS DOCUMENTATION

This reference manual was 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