PySirix

Release 0.5.1

Moshe Uminer

Aug 17, 2023

CONTENTS:

CHAPTER

ONE

PYSIRIX PACKAGE

1.1 Module contents

class pysirix.**Commit**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

author: str

commitMessage: str

revision: int

revisionTimestamp: str

class pysirix.**DBType**(*value*)

Bases: `enum.Enum`

This Enum class defines the possible database (and resource) types

JSON = 'application/json'

XML = 'application/xml'

class pysirix.**Database**(*database_name: str*, *database_type:* pysirix.constants.DBType, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

Bases: `object`

__init__(*database_name: str*, *database_type:* pysirix.constants.DBType, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

Database access class

This class allows for manipulation of a database

Parameters

- **database_name** – the name of the database to access, or create if it does not yet exist
- **database_type** – the type of the database being accessed, or to be created if the database does not yet exist
- **client** – the `SyncClient` or `AsyncClient` instance to use for network requests
- **auth** – the `Auth` that keeps the client authenticated. It is referenced to ensure that it never goes out of scope

create() → Union[None, Awaitable[None]]

Create a database with the name and type of this `Database` instance.

delete() → Optional[Awaitable[None]]

Delete the database with the name of this `Database` instance.

get_database_info() → Union[Awaitable[Dict], Dict]

Get information about the resources of this database. Raises a `SirixServerError` error if the database does not exist.

Returns a `dict` with the name, type, and resources (as a `list` of `str`) of this database.

Raises `SirixServerError`.

json_store(*name: str*, *root: str = ''*)

Returns a `JsonStoreSync` or `JsonStoreAsync` instance, depending or whether `sirix_sync()` or `sirix_async()` was used for initialization.

Parameters

- **name** – the resource name for the store.
- **root** – where the store is located in the resource.

Returns an instance of `JsonStoreSync` or `JsonStoreAsync`.

resource(*resource_name: str*)

Returns a `resource` instance.

Parameters **resource_name** – the name of the resource to access

Returns an instance of `Resource`.

class pysirix.**DeleteDiff**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

depth: int

deweyID: str

nodeKey: int

class pysirix.**Insert**(*value*)

Bases: `enum.Enum`

This Enum class defines the possible options for a resource update

CHILD = 'asFirstChild'

LEFT = 'asLeftSibling'

REPLACE = 'replace'

RIGHT = 'asRightSibling'

class pysirix.**InsertDiff**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

data: *pysirix.info.DataType*

depth: int

deweyID: str

insertPosition: *pysirix.info.InsertPosition*

insertPositionNodeKey: int

nodeKey: int

type: str

`class pysirix.`**`JsonStoreAsync`**`(`*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient*,* pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*`)`

Bases: *`pysirix.json_store.JsonStoreBase`*

This class is a convenient abstraction over the resource entities exposed by SirixDB. As such, there is no JsonStore on the SirixDB server, only the underlying resource is stored.

This class is for storing many distinct, JSON objects in a single resource, where the objects/records store data of similar type. As such, it's usage parallels a that of a document store, and an object is an abstraction similar to a single document in such a store.

db_name

db_type

`async` **`find_all`**`(`*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*`)` → List[*pysirix.types.QueryResult*]

Finds and returns all records where the values of `query_dict` match the corresponding values the record.

`projection` can optionally be used to retrieve only certain fields of the matching records.

By default, the node_key of of each record is returned as a `nodeKey` field in the record. The `node_key` parameter controls this behavior.

Parameters

- **`query_dict`** – a `dict` with which to query the records.
- **`projection`** – a `list` of field names to return for the matching records.
- **`node_key`** – a `bool` determining whether or not to return a `nodeKey` field containing the nodeKey of the record.
- **`hash`** – a `bool` determining whether or not to return a `hash` field containing the hash of the record.
- **`revision`** – the revision to search, defaults to latest. May be an integer or a `datetime` instance
- **`time_axis_shift`** – specify fetching the most or least recent existing revision of the record
- **`start_result_index`** – index of first result to return.
- **`end_result_index`** – index of last result to return.

Returns a `list` of *`QueryResult`* records matching the query.

`async` **`find_by_key`**`(`*node_key: Optional[int]*, *revision: Optional[Union[int, datetime.datetime]] = None*`)`

Parameters

- **`node_key`** – the nodeKey of the record to read
- **`revision`** – the revision to search, defaults to latest. May be an integer or a `datetime` instance

Returns

async history(*node_key: int*, *subtree: bool = True*, *revision: Optional[Union[int, datetime.datetime]] = None*) → Union[List[*pysirix.types.SubtreeRevision*], List[*pysirix.types.Revision*]]

This method returns the history of a node in the resource.

Parameters

- **node_key** – the root of the subtree whose history should be returned. Defaults to document root.
- **subtree** – whether to account for changes in the subtree of the given node. Defaults to `True`.
- **revision** – the revision in which the node with the given `node_key` exists (if it does not exist currently). Defaults to the latest revision. May be an integer or a `datetime` instance

Returns If `subtree` is `True`, a list of `pysirix.types.SubtreeRevision`. Else, a list of `pysirix.types.RevisionType`.

async history_embed(*node_key: int*, *revision: Optional[Union[int, datetime.datetime]] = None*) → List[*pysirix.types.QueryResult*]

name

async resource_history() → List[*pysirix.types.Commit*]

This method returns the entire history of a resource.

Returns a list of `Commit`.

class pysirix.JsonStoreSync(*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*)

Bases: `pysirix.json_store.JsonStoreBase`

This class is a convenient abstraction over the resource entities exposed by SirixDB. As such, there is no JsonStore on the SirixDB server, only the underlying resource is stored.

This class is for storing many distinct, JSON objects in a single resource, where the objects/records store data of similar type. As such, it's usage parallels a that of a document store, and an object is an abstraction similar to a single document in such a store.

db_name

db_type

find_all(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*) → List[*pysirix.types.QueryResult*]

Finds and returns all records where the values of `query_dict` match the corresponding values the record.

`projection` can optionally be used to retrieve only certain fields of the matching records.

By default, the node_key of of each record is returned as a `nodeKey` field in the record. The `node_key` parameter controls this behavior.

Parameters

- **query_dict** – a `dict` with which to query the records.
- **projection** – a `list` of field names to return for the matching records.
- **node_key** – a `bool` determining whether or not to return a `nodeKey` field containing the nodeKey of the record.
- **hash** – a `bool` determining whether or not to return a `hash` field containing the hash of the record.

- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance
- **time_axis_shift** – specify fetching the most or least recent existing revision of the record
- **start_result_index** – index of first result to return.
- **end_result_index** – index of last result to return.

Returns a `list` of `QueryResult` records matching the query.

`find_by_key`(*node_key: Optional[int]*, *revision: Optional[Union[int, datetime.datetime]] = None*)

Parameters

- **node_key** – the nodeKey of the record to read
- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance

Returns

`history`(*node_key: int*, *subtree: bool = True*, *revision: Optional[Union[int, datetime.datetime]] = None*) → Union[List[*pysirix.types.SubtreeRevision*], List[*pysirix.types.Revision*]]

This method returns the history of a node in the resource.

Parameters

- **node_key** – the root of the subtree whose history should be returned. Defaults to document root.
- **subtree** – whether to account for changes in the subtree of the given node. Defaults to `True`.
- **revision** – the revision in which the node with the given `node_key` exists (if it does not exist currently). Defaults to the latest revision. May be an integer or a `datetime` instance

Returns If `subtree` is `True`, a list of `pysirix.types.SubtreeRevision`. Else, a list of `pysirix.types.RevisionType`.

`history_embed`(*node_key: int*, *revision: Optional[Union[int, datetime.datetime]] = None*) → List[*pysirix.types.QueryResult*]

`name`

`resource_history`() → List[*pysirix.types.Commit*]

This method returns the entire history of a resource.

Returns a list of `Commit`.

`class pysirix.MetaNode`(**args*, ***kwargs*)

Bases: `dict`

`key` is provided only if `type` is `pysirix.info.NodeType` `OBJECT_KEY`.

`value` is of type `List[MetaNode]` if `metadata.type` is `OBJECT` or `ARRAY`, however, if `metadata.childCount` is 0, then `value` is an emtpy `dict`, or an empty `list`, depending on whether `metadata.type` is `OBJECT` or `ARRAY`.

`value` is of type `MetaNode` if `metadata.type` is `OBJECT_KEY`.

`value` is a `str` if `metadata.type` is `OBJECT_STRING_VALUE` or `STRING_VALUE`.

`value` is an `int` or `float` if `metadata.type` == `OBJECT_NUMBER_VALUE` or `NUMBER_VALUE`.

`value` is a `bool` if `metadata.type` is `OBJECT_BOOLEAN_VALUE` or `BOOLEAN_VALUE`.

value is None if metadata.type is OBJECT_NULL_VALUE or NULL_VALUE.

key: str

metadata: *pysirix.types.Metadata*

value: Optional[Union[List[Iterable[*pysirix.types.MetaNode*]], Iterable[*pysirix.types.MetaNode*], str, int, float, bool]]

class pysirix.**Metadata**(**args*, ***kwargs*)

Bases: dict

descendantCount and childCount are provided only where type is *pysirix.info.NodeType* OBJECT or ARRAY.

childCount: int

descendantCount: int

hash: int

nodeKey: int

type: *pysirix.info.NodeType*

class pysirix.**QueryResult**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

revision: Union[Dict, List]

revisionNumber: int

revisionTimestamp: str

class pysirix.**ReplaceDiff**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

data: str

nodeKey: int

type: *pysirix.info.DataType*

class pysirix.**Resource**(*db_name: str*, *db_type:* pysirix.constants.DBType, *resource_name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

Bases: object

__init__(*db_name: str*, *db_type:* pysirix.constants.DBType, *resource_name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

Resource access class.

This class allows for manipulation of a resource

Parameters

- **db_name** – the name of the database this resource belongs to.
- **db_type** – the type of data the database can hold.
- **resource_name** – the name of the resource being accessed, or to be created if the resource does not yet exist
- **client** – the SyncClient or AsyncClient instance to use for network requests

- **auth** – the Auth that keeps the client authenticated. It is referenced to ensure that it never goes out of scope

create(*data: Union[str, Dict, xml.etree.ElementTree.Element]*, *hash_type: str = 'ROLLING'*)

Parameters **data** – the data with which to initialize the resource. May be an instance of `dict`, or an instance of `xml.etree.ElementTree.Element` (depending on the database type), or a `str` of properly formed json or xml.

delete(*node_id: Optional[int]*, *etag: Optional[str]*) → Union[None, Awaitable[None]]

Delete a node in a resource, or, if `node_id` is specified as `None`, delete the entire resource.

Parameters

- **node_id** – an `int` corresponding to the node to delete. Should be specified as none to delete the entire resource.
- **etag** – the `etag` of the node to delete. This can be fetched using the py:method`get_etag` method. If `etag` is specified as `None`, then the `etag` will be fetched and provided implicitly.

diff(*first_revision: Union[int, datetime.datetime]*, *second_revision: Union[int, datetime.datetime]*, *node_id: Optional[int] = None*, *max_depth: Optional[int] = None*)

exists()

Sends a `head` request to determine whether or not this store/resource already exists.

Returns a `bool` corresponding to the existence of the store.

get_etag(*node_id: int*) → Union[str, Awaitable[str]]

Get the ETag of a given node.

Parameters **node_id** – the nodeKey corresponding to which the ETag should be returned.

Returns a `str` ETag.

history() → List[*pysirix.types.Commit*]

Get a `list` of all commits/revision of this resource.

Returns a `list` of `dict` of the form *`pysirix.Commit`*.

query(*query: str*, *start_result_seq_index: Optional[int] = None*, *end_result_seq_index: Optional[int] = None*)

Execute a custom query on this resource. The `start_result_seq_index` and `end_result_seq_index` can be used for pagination.

Parameters

- **query** – the query `str` to execute.
- **start_result_seq_index** – the first index of the results from which to return, defaults to first.
- **end_result_seq_index** – the last index of the results to return, defaults to last.

Returns the query result.

read(*node_id: Optional[int]*, *revision: Optional[Union[int, datetime.datetime, Tuple[Union[int, datetime.datetime], Union[int, datetime.datetime]]]] = None*, *max_level: Optional[int] = None*, *top_level_limit: Optional[int] = None*, *top_level_skip_last_node: Optional[int] = None*) → Union[dict, xml.etree.ElementTree.Element, Awaitable[Union[dict, xml.etree.ElementTree.Element]]]

Read the node (and its sub-nodes) corresponding to `node_id`.

Parameters

- **`node_id`** – the nodeKey corresponding to the node to read, if `None`, the entire resource is read.
- **`revision`** – the revision to read from, defaults to latest.
- **`max_level`** – the maximum depth for reading sub-nodes, defaults to latest.
- **`top_level_limit`** – the maximum number of top level nodes to return (used for paging).
- **`top_level_skip_last_node`** – the last nodeId to skip (used for paging).

Returns either a `dict` or an instance of `xml.etree.ElementTree.Element`, depending on the database type of this resource.

`read_with_metadata`(*node_id: Optional[int]*, *revision: Optional[Union[int, datetime.datetime, Tuple[Union[int, datetime.datetime], Union[int, datetime.datetime]]]] = None*, *meta_type:* pysirix.constants.MetadataType *= MetadataType.ALL*, *max_level: Optional[int] = None*, *top_level_limit: Optional[int] = None*, *top_level_skip_last_node: Optional[int] = None*)

Read the node (and its sub-nodes) corresponding to `node_id`, with metadata for each node.

Parameters

- **`node_id`** – the nodeKey corresponding to the node to read, if `None`, the entire resource is read.
- **`revision`** – the revision to read from, defaults to latest.
- **`meta_type`** – the type of metadata to return, defaults to all.
- **`max_level`** – the maximum depth for reading sub-nodes, defaults to latest.
- **`top_level_limit`** – the maximum number of top level nodes to return (used for paging).
- **`top_level_skip_last_node`** – the last nodeId to skip (used for paging).

Returns

`update`(*node_id: int*, *data: Union[str, xml.etree.ElementTree.Element, Dict]*, *insert:* pysirix.constants.Insert *= Insert.CHILD*, *etag: Optional[str] = None*) → Union[str, Awaitable[str]]

Update a resource.

Parameters

- **`node_id`** – the nodekey in reference to which the update should be performed.
- **`data`** – the updated data, can be of type `str`, `dict`, or `xml.etree.ElementTree.Element`
- **`insert`** – the position of the update in relation to the node referenced by node_id.
- **`etag`** – the ETag of the node referenced by node_id.

`class pysirix.`**`Revision`**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

`revision: Optional[Union[List, Dict, str, int, float]]`

`revisionNumber: int`

`revisionTimestamp: str`

`class pysirix.`**`Sirix`**(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*)

Bases: `object`

`__init__`(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*)
SirixDB access class. This class is the entrypoint for manipulating data with SirixDB.

Parameters

- **username** – the username registered with keycloak for this application.
- **password** – the password registered with keycloak for this application.
- **client** – the `httpx.Client` or `httpx.AsyncClient` to use.

`authenticate`()
Call the authenticate endpoint. Must be called before any other calls are made. This is done internally by *sirix_sync()* or by *sirix_async()*.

`database`(*database_name: str*, *database_type:* pysirix.constants.DBType)
Returns a *Database* instance.

Parameters

- **database_name** – the name of the database to access.
- **database_type** – the type of the database to access.

`delete_all`() → Optional[Coroutine]
Deletes all databases and resources in the SirixDB server. Be careful!

`dispose`()
Remove the authentication timer.

`get_info`(*resources: bool = True*) → Union[Coroutine, List[Dict[str, str]]]
Returns a list of database names and types, and (optionally) a list their resources as well.

Parameters **resources** – whether or not to include resource information

Returns a `list` of `dicts`, where each `dict` has a `name` field, a `type` field, and (if resources is `True`) a `resources` field (containing a `list` of names).

`query`(*query: str*, *start_result_seq_index: Optional[int] = None*, *end_result_seq_index: Optional[int] = None*)
Execute a custom query on SirixDB. Unlike the query method on *Resource*, queries executed with this method potentially access the entirety of the SirixDB server. The `start_result_seq_index` and `end_result_seq_index` can be used for pagination.

Parameters

- **query** – the query `str` to execute.
- **start_result_seq_index** – the first index of the results from which to return, defaults to first.
- **end_result_seq_index** – the last index of the results to return, defaults to last.

Returns the query result.

exception pysirix.`SirixServerError`(*message: str*, ***, *request: Request*, *response: Response*)
Bases: `httpx.HTTPStatusError`

class pysirix.`TimeAxisShift`(*value*)
Bases: `enum.Enum`

An enumeration.

`latest = 1`

`none = 0`

`oldest = -1`

class `pysirix.UpdateDiff(*args, **kwargs)`

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

nodeKey: int

type: *pysirix.info.DataType*

value: Optional[Union[str, int, float, bool]]

async `pysirix.sirix_async`(*username: str*, *password: str*, *client: httpx.AsyncClient*) → *pysirix.sirix.Sirix*

Parameters

- **username** – the username registered with keycloak for this application.
- **password** – the password registered with keycloak for this application.
- **client** – an `httpx.AsyncClient` instance. You should instantiate the instance with the `base_url` param as the url for the sirix database.

`pysirix.sirix_sync`(*username: str*, *password: str*, *client: httpx.Client*) → *pysirix.sirix.Sirix*

Parameters

- **username** – the username registered with keycloak for this application.
- **password** – the password registered with keycloak for this application.
- **client** – an `httpx.Client` instance. You should instantiate the instance with the `base_url` param as the url for the sirix database.

1.2 Submodules

1.3 pysirix.sirix module

class `pysirix.sirix.Sirix`(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*)

`__init__`(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*)

SirixDB access class. This class is the entrypoint for manipulating data with SirixDB.

Parameters

- **username** – the username registered with keycloak for this application.
- **password** – the password registered with keycloak for this application.
- **client** – the `httpx.Client` or `httpx.AsyncClient` to use.

`authenticate()`

Call the authenticate endpoint. Must be called before any other calls are made. This is done internally by `sirix_sync()` or by `sirix_async()`.

`database`(*database_name: str*, *database_type:* pysirix.constants.DBType)

Returns a `Database` instance.

Parameters

- **database_name** – the name of the database to access.
- **database_type** – the type of the database to access.

`delete_all()` → Optional[Coroutine]
Deletes all databases and resources in the SirixDB server. Be careful!

`dispose()`
Remove the authentication timer.

`get_info`(*resources: bool = True*) → Union[Coroutine, List[Dict[str, str]]]
Returns a list of database names and types, and (optionally) a list their resources as well.

Parameters **resources** – whether or not to include resource information

Returns a `list` of `dicts`, where each `dict` has a `name` field, a `type` field, and (if resources is `True`) a `resources` field (containing a `list` of names).

`query`(*query: str*, *start_result_seq_index: Optional[int] = None*, *end_result_seq_index: Optional[int] = None*)
Execute a custom query on SirixDB. Unlike the query method on `Resource`, queries executed with this method potentially access the entirety of the SirixDB server. The `start_result_seq_index` and `end_result_seq_index` can be used for pagination.

Parameters

- **query** – the query `str` to execute.
- **start_result_seq_index** – the first index of the results from which to return, defaults to first.
- **end_result_seq_index** – the last index of the results to return, defaults to last.

Returns the query result.

1.4 pysirix.database module

`class pysirix.database.Database`(*database_name: str*, *database_type:* pysirix.constants.DBType, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

`__init__`(*database_name: str*, *database_type:* pysirix.constants.DBType, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)
Database access class

This class allows for manipulation of a database

Parameters

- **database_name** – the name of the database to access, or create if it does not yet exist
- **database_type** – the type of the database being accessed, or to be created if the database does not yet exist
- **client** – the `SyncClient` or `AsyncClient` instance to use for network requests
- **auth** – the `Auth` that keeps the client authenticated. It is referenced to ensure that it never goes out of scope

`create()` → Union[None, Awaitable[None]]
Create a database with the name and type of this `Database` instance.

delete() → Optional[Awaitable[None]]

Delete the database with the name of this *Database* instance.

get_database_info() → Union[Awaitable[Dict], Dict]

Get information about the resources of this database. Raises a `SirixServerError` error if the database does not exist.

Returns a `dict` with the name, type, and resources (as a `list` of `str`) of this database.

Raises `SirixServerError`.

json_store(*name: str*, *root: str = ''*)

Returns a `JsonStoreSync` or `JsonStoreAsync` instance, depending or whether `sirix_sync()` or `sirix_async()` was used for initialization.

Parameters

- **name** – the resource name for the store.
- **root** – where the store is located in the resource.

Returns an instance of `JsonStoreSync` or `JsonStoreAsync`.

resource(*resource_name: str*)

Returns a *resource* instance.

Parameters **resource_name** – the name of the resource to access

Returns an instance of `Resource`.

1.5 pysirix.resource module

class pysirix.resource.**Resource**(*db_name: str*, *db_type:* pysirix.constants.DBType, *resource_name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

__init__(*db_name: str*, *db_type:* pysirix.constants.DBType, *resource_name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth)

Resource access class.

This class allows for manipulation of a resource

Parameters

- **db_name** – the name of the database this resource belongs to.
- **db_type** – the type of data the database can hold.
- **resource_name** – the name of the resource being accessed, or to be created if the resource does not yet exist
- **client** – the `SyncClient` or `AsyncClient` instance to use for network requests
- **auth** – the `Auth` that keeps the client authenticated. It is referenced to ensure that it never goes out of scope

`static _build_read_params`(*node_id: Optional[int]*, *revision: Optional[Union[int, datetime.datetime, Tuple[Union[int, datetime.datetime], Union[int, datetime.datetime]]]] = None*, *max_level: Optional[int] = None*, *top_level_limit: Optional[int] = None*, *top_level_skip_last_node: Optional[int] = None*) → Dict[str, Union[str, int]]

Helper method to build a parameters `dict` for reading a resource.

`create`(*data: Union[str, Dict, xml.etree.ElementTree.Element]*, *hash_type: str = 'ROLLING'*)

Parameters **`data`** – the data with which to initialize the resource. May be an instance of `dict`, or an instance of `xml.etree.ElementTree.Element` (depending on the database type), or a `str` of properly formed json or xml.

`delete`(*node_id: Optional[int]*, *etag: Optional[str]*) → Union[None, Awaitable[None]]

Delete a node in a resource, or, if `node_id` is specified as `None`, delete the entire resource.

Parameters

- **`node_id`** – an `int` corresponding to the node to delete. Should be specified as none to delete the entire resource.
- **`etag`** – the `etag` of the node to delete. This can be fetched using the py:method`get_etag` method. If `etag` is specified as `None`, then the `etag` will be fetched and provided implicitly.

`diff`(*first_revision: Union[int, datetime.datetime]*, *second_revision: Union[int, datetime.datetime]*, *node_id: Optional[int] = None*, *max_depth: Optional[int] = None*)

`exists`()

Sends a `head` request to determine whether or not this store/resource already exists.

Returns a `bool` corresponding to the existence of the store.

`get_etag`(*node_id: int*) → Union[str, Awaitable[str]]

Get the ETag of a given node.

Parameters **`node_id`** – the nodeKey corresponding to which the ETag should be returned.

Returns a `str` ETag.

`history`() → List[*pysirix.types.Commit*]

Get a `list` of all commits/revision of this resource.

Returns a `list` of `dict` of the form `pysirix.Commit`.

`query`(*query: str*, *start_result_seq_index: Optional[int] = None*, *end_result_seq_index: Optional[int] = None*)

Execute a custom query on this resource. The `start_result_seq_index` and `end_result_seq_index` can be used for pagination.

Parameters

- **`query`** – the query `str` to execute.
- **`start_result_seq_index`** – the first index of the results from which to return, defaults to first.
- **`end_result_seq_index`** – the last index of the results to return, defaults to last.

Returns the query result.

read(*node_id: Optional[int]*, *revision: Optional[Union[int, datetime.datetime, Tuple[Union[int, datetime.datetime], Union[int, datetime.datetime]]]] = None*, *max_level: Optional[int] = None*, *top_level_limit: Optional[int] = None*, *top_level_skip_last_node: Optional[int] = None*) → Union[dict, xml.etree.ElementTree.Element, Awaitable[Union[dict, xml.etree.ElementTree.Element]]]

Read the node (and its sub-nodes) corresponding to `node_id`.

Parameters

- **node_id** – the nodeKey corresponding to the node to read, if `None`, the entire resource is read.
- **revision** – the revision to read from, defaults to latest.
- **max_level** – the maximum depth for reading sub-nodes, defaults to latest.
- **top_level_limit** – the maximum number of top level nodes to return (used for paging).
- **top_level_skip_last_node** – the last nodeId to skip (used for paging).

Returns either a `dict` or an instance of `xml.etree.ElementTree.Element`, depending on the database type of this resource.

read_with_metadata(*node_id: Optional[int]*, *revision: Optional[Union[int, datetime.datetime, Tuple[Union[int, datetime.datetime], Union[int, datetime.datetime]]]] = None*, *meta_type:* pysirix.constants.MetadataType *= MetadataType.ALL*, *max_level: Optional[int] = None*, *top_level_limit: Optional[int] = None*, *top_level_skip_last_node: Optional[int] = None*)

Read the node (and its sub-nodes) corresponding to `node_id`, with metadata for each node.

Parameters

- **node_id** – the nodeKey corresponding to the node to read, if `None`, the entire resource is read.
- **revision** – the revision to read from, defaults to latest.
- **meta_type** – the type of metadata to return, defaults to all.
- **max_level** – the maximum depth for reading sub-nodes, defaults to latest.
- **top_level_limit** – the maximum number of top level nodes to return (used for paging).
- **top_level_skip_last_node** – the last nodeId to skip (used for paging).

Returns

update(*node_id: int*, *data: Union[str, xml.etree.ElementTree.Element, Dict]*, *insert:* pysirix.constants.Insert *= Insert.CHILD*, *etag: Optional[str] = None*) → Union[str, Awaitable[str]]

Update a resource.

Parameters

- **node_id** – the nodekey in reference to which the update should be performed.
- **data** – the updated data, can be of type `str`, `dict`, or `xml.etree.ElementTree.Element`
- **insert** – the position of the update in relation to the node referenced by node_id.
- **etag** – the ETag of the node referenced by node_id.

1.6 pysirix.json_store module

class pysirix.json_store.**JsonStoreAsync**(*db_name: str*, *name: str*, *client: Union[pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient]*, *auth: pysirix.auth.Auth*, *root: str = ''*)

Bases: *pysirix.json_store.JsonStoreBase*

This class is a convenient abstraction over the resource entities exposed by SirixDB. As such, there is no JsonStore on the SirixDB server, only the underlying resource is stored.

This class is for storing many distinct, JSON objects in a single resource, where the objects/records store data of similar type. As such, it's usage parallels a that of a document store, and an object is an abstraction similar to a single document in such a store.

__init__(*db_name: str*, *name: str*, *client: Union[pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient]*, *auth: pysirix.auth.Auth*, *root: str = ''*)

_abc_impl = <_abc_data object>

_auth

_client

_prepare_find_all(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift: pysirix.constants.TimeAxisShift = TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*)

create(*data: str = '[]'*) → Union[str, Awaitable[str]]

Creates the store, will overwrite the store if it already exists.

Parameters **data** – data with which to initialize the store

Returns will return the string "[]". If in async mode, an awaitable that resolves this string.

db_name

db_type

delete_field(*query_dict: Dict*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **query_dict** – a `dict` of field names and their values to match against
- **fields** – the keys of the fields of the records to delete

Returns

delete_fields_by_key(*node_key: int*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **node_key** – the nodeKey of the record to delete
- **fields** – the keys of the fields of the record to delete

Returns

delete_records(*query_dict: Dict*) → Union[str, Awaitable[str]]

Parameters **`query_dict`** – a `dict` of field names and their values to match against

Returns

`exists``()` → Union[bool, Awaitable[bool]]

Sends a `head` request to determine whether or not this store/resource already exists.

Returns a `bool` corresponding to the existence of the store.

`async` **`find_all`**`(`*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift: pysirix.constants.TimeAxisShift = TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*`)` → List[*pysirix.types.QueryResult*]

Finds and returns all records where the values of `query_dict` match the corresponding values the record.

`projection` can optionally be used to retrieve only certain fields of the matching records.

By default, the node_key of of each record is returned as a `nodeKey` field in the record. The `node_key` parameter controls this behavior.

Parameters

- **`query_dict`** – a `dict` with which to query the records.
- **`projection`** – a `list` of field names to return for the matching records.
- **`node_key`** – a `bool` determining whether or not to return a `nodeKey` field containing the nodeKey of the record.
- **`hash`** – a `bool` determining whether or not to return a `hash` field containing the hash of the record.
- **`revision`** – the revision to search, defaults to latest. May be an integer or a `datetime` instance
- **`time_axis_shift`** – specify fetching the most or least recent existing revision of the record
- **`start_result_index`** – index of first result to return.
- **`end_result_index`** – index of last result to return.

Returns a `list` of `QueryResult` records matching the query.

`async` **`find_by_key`**`(`*node_key: Optional[int]*, *revision: Optional[Union[int, datetime.datetime]] = None*`)`

Parameters

- **`node_key`** – the nodeKey of the record to read
- **`revision`** – the revision to search, defaults to latest. May be an integer or a `datetime` instance

Returns

`find_one``(`*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift: pysirix.constants.TimeAxisShift = TimeAxisShift.none*`)` → List[*pysirix.types.QueryResult*]

This method is the same as `find_many()`, except that this method will only return the first result,
by way of passing `0` to that method's `start_result_index`, and `end_result_index` parameters.

Parameters

- **`query_dict`** –

- **projection** –
- **revision** –
- **node_key** –
- **hash** –
- **time_axis_shift** –

Returns

async **history**(*node_key: int*, *subtree: bool = True*, *revision: Optional[Union[int, datetime.datetime]] = None*) → Union[List[*pysirix.types.SubtreeRevision*], List[*pysirix.types.Revision*]]

This method returns the history of a node in the resource.

Parameters

- **node_key** – the root of the subtree whose history should be returned. Defaults to document root.
- **subtree** – whether to account for changes in the subtree of the given node. Defaults to `True`.
- **revision** – the revision in which the node with the given `node_key` exists (if it does not exist currently). Defaults to the latest revision. May be an integer or a `datetime` instance

Returns If `subtree` is `True`, a list of *`pysirix.types.SubtreeRevision`*. Else, a list of `pysirix.types.RevisionType`.

async **history_embed**(*node_key: int*, *revision: Optional[Union[int, datetime.datetime]] = None*) → List[*pysirix.types.QueryResult*]

insert_many(*insert_list: Union[str, List[Dict]]*) → Union[str, Awaitable[str]]

Inserts a list of records into the store. New records are added at the the tail of the store.

Parameters insert_list – either a JSON string of `list` of `dicts`, or a `list` that can be converted to JSON

Returns a `str` "{rest: []}" or an `Awaitable[str]` resolving to this string.

insert_one(*insert_dict: Union[str, Dict]*) → Union[str, Awaitable[str]]

Inserts a single record into the store. New records are added at the tail of the store.

Parameters insert_dict – either a JSON string of a `dict`, or a `dict` that can be converted to JSON.

Returns an emtpy `str` or an empty `Awaitable[str]`.

name

async **resource_history**() → List[*pysirix.types.Commit*]

This method returns the entire history of a resource.

Returns a list of `Commit`.

update_by_key(*node_key: int*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **node_key** – the nodeKey of the record to update
- **update_dict** – a dict of keys and matching values to replace in the given record

- **`upsert`** – whether to insert if the field does not already exist

Returns

`update_many`(*query_dict: Dict*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **`query_dict`** – a `dict` of field names and their values to match against
- **`update_dict`** – a dict of keys and matching values to replace in the selected record
- **`upsert`** – whether to insert if the field does not already exist

Returns

`class pysirix.json_store.`**`JsonStoreBase`**(*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*)

Bases: `abc.ABC`

`__init__`(*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*)

`_abc_impl = <_abc_data object>`

`_auth`

`_client`

`_prepare_find_all`(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*)

`create`(*data: str = '[]'*) → Union[str, Awaitable[str]]

Creates the store, will overwrite the store if it already exists.

Parameters **`data`** – data with which to initialize the store

Returns will return the string "[]". If in async mode, an awaitable that resolves this string.

`db_name`

`db_type`

`delete_field`(*query_dict: Dict*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **`query_dict`** – a `dict` of field names and their values to match against
- **`fields`** – the keys of the fields of the records to delete

Returns

`delete_fields_by_key`(*node_key: int*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **`node_key`** – the nodeKey of the record to delete

- **fields** – the keys of the fields of the record to delete

Returns

delete_records(*query_dict: Dict*) → Union[str, Awaitable[str]]

Parameters **query_dict** – a `dict` of field names and their values to match against

Returns

exists() → Union[bool, Awaitable[bool]]

Sends a `head` request to determine whether or not this store/resource already exists.

Returns a `bool` corresponding to the existence of the store.

find_all(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*) → List[*pysirix.types.QueryResult*]

Finds and returns all records where the values of `query_dict` match the corresponding values the record.

`projection` can optionally be used to retrieve only certain fields of the matching records.

By default, the node_key of of each record is returned as a `nodeKey` field in the record. The `node_key` parameter controls this behavior.

Parameters

- **query_dict** – a `dict` with which to query the records.
- **projection** – a `list` of field names to return for the matching records.
- **node_key** – a `bool` determining whether or not to return a `nodeKey` field containing the nodeKey of the record.
- **hash** – a `bool` determining whether or not to return a `hash` field containing the hash of the record.
- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance
- **time_axis_shift** – specify fetching the most or least recent existing revision of the record
- **start_result_index** – index of first result to return.
- **end_result_index** – index of last result to return.

Returns a `list` of `QueryResult` records matching the query.

find_by_key(*node_key: Optional[int]*, *revision: Optional[Union[int, datetime.datetime]] = None*)

Parameters

- **node_key** – the nodeKey of the record to read
- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance

Returns

find_one(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*) → List[*pysirix.types.QueryResult*]

This method is the same as `find_many()`, except that this method will only return the first result,
by way of passing `0` to that method's `start_result_index`, and `end_result_index` parameters.

Parameters

- **`query_dict`** –
- **`projection`** –
- **`revision`** –
- **`node_key`** –
- **`hash`** –
- **`time_axis_shift`** –

Returns

`history`(*node_key: int*, *subtree: bool = True*, *revision: Optional[Union[int, datetime.datetime]] = None*)
This method returns the history of a node in the resource.

Parameters

- **`node_key`** – the root of the subtree whose history should be returned. Defaults to document root.
- **`subtree`** – whether to account for changes in the subtree of the given node. Defaults to `True`.
- **`revision`** – the revision in which the node with the given `node_key` exists (if it does not exist currently). Defaults to the latest revision. May be an integer or a `datetime` instance

Returns If `subtree` is `True`, a list of *`pysirix.types.SubtreeRevision`*. Else, a list of `pysirix.types.RevisionType`.

`history_embed`(*node_key: int*, *revision: Optional[Union[int, datetime.datetime]] = None*)

`insert_many`(*insert_list: Union[str, List[Dict]]*) → Union[str, Awaitable[str]]
Inserts a list of records into the store. New records are added at the the tail of the store.

Parameters **`insert_list`** – either a JSON string of `list` of `dicts`, or a `list` that can be converted to JSON

Returns a `str` "{rest: []}" or an `Awaitable[str]` resolving to this string.

`insert_one`(*insert_dict: Union[str, Dict]*) → Union[str, Awaitable[str]]
Inserts a single record into the store. New records are added at the tail of the store.

Parameters **`insert_dict`** – either a JSON string of a `dict`, or a `dict` that can be converted to JSON.

Returns an emtpy `str` or an empty `Awaitable[str]`.

`name`

`resource_history`() → Union[List[*pysirix.types.Commit*], Awaitable[List[*pysirix.types.Commit*]]]
This method returns the entire history of a resource.

Returns a list of `Commit`.

`update_by_key`(*node_key: int*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **node_key** – the nodeKey of the record to update
- **update_dict** – a dict of keys and matching values to replace in the given record
- **upsert** – whether to insert if the field does not already exist

Returns

update_many(*query_dict: Dict*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **query_dict** – a `dict` of field names and their values to match against
- **update_dict** – a dict of keys and matching values to replace in the selected record
- **upsert** – whether to insert if the field does not already exist

Returns

class pysirix.json_store.**JsonStoreSync**(*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*)

Bases: *pysirix.json_store.JsonStoreBase*

This class is a convenient abstraction over the resource entities exposed by SirixDB. As such, there is no JsonStore on the SirixDB server, only the underlying resource is stored.

This class is for storing many distinct, JSON objects in a single resource, where the objects/records store data of similar type. As such, it's usage parallels a that of a document store, and an object is an abstraction similar to a single document in such a store.

__init__(*db_name: str*, *name: str*, *client: Union[*pysirix.sync_client.SyncClient, pysirix.async_client.AsyncClient*]*, *auth:* pysirix.auth.Auth, *root: str = ''*)

_abc_impl = <_abc_data object>

_auth

_client

_prepare_find_all(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift:* pysirix.constants.TimeAxisShift *= TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*)

create(*data: str = '[]'*) → Union[str, Awaitable[str]]

Creates the store, will overwrite the store if it already exists.

Parameters **data** – data with which to initialize the store

Returns will return the string "[]". If in async mode, an awaitable that resolves this string.

db_name

db_type

delete_field(*query_dict: Dict*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **query_dict** – a `dict` of field names and their values to match against

- **fields** – the keys of the fields of the records to delete

Returns

delete_fields_by_key(*node_key: int*, *fields: List[str]*) → Union[str, Awaitable[str]]

Parameters

- **node_key** – the nodeKey of the record to delete
- **fields** – the keys of the fields of the record to delete

Returns

delete_records(*query_dict: Dict*) → Union[str, Awaitable[str]]

Parameters **query_dict** – a `dict` of field names and their values to match against

Returns

exists() → Union[bool, Awaitable[bool]]

Sends a `head` request to determine whether or not this store/resource already exists.

Returns a `bool` corresponding to the existence of the store.

find_all(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift: pysirix.constants.TimeAxisShift = TimeAxisShift.none*, *start_result_index: Optional[int] = None*, *end_result_index: Optional[int] = None*) → List[*pysirix.types.QueryResult*]

Finds and returns all records where the values of `query_dict` match the corresponding values the record.

`projection` can optionally be used to retrieve only certain fields of the matching records.

By default, the node_key of of each record is returned as a `nodeKey` field in the record. The `node_key` parameter controls this behavior.

Parameters

- **query_dict** – a `dict` with which to query the records.
- **projection** – a `list` of field names to return for the matching records.
- **node_key** – a `bool` determining whether or not to return a `nodeKey` field containing the nodeKey of the record.
- **hash** – a `bool` determining whether or not to return a `hash` field containing the hash of the record.
- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance
- **time_axis_shift** – specify fetching the most or least recent existing revision of the record
- **start_result_index** – index of first result to return.
- **end_result_index** – index of last result to return.

Returns a `list` of `QueryResult` records matching the query.

find_by_key(*node_key: Optional[int]*, *revision: Optional[Union[int, datetime.datetime]] = None*)

Parameters

- **node_key** – the nodeKey of the record to read

- **revision** – the revision to search, defaults to latest. May be an integer or a `datetime` instance

Returns

find_one(*query_dict: Dict*, *projection: Optional[List[str]] = None*, *revision: Optional[Union[int, datetime.datetime]] = None*, *node_key: bool = True*, *hash: bool = False*, *time_axis_shift: pysirix.constants.TimeAxisShift = TimeAxisShift.none*) → List[*pysirix.types.QueryResult*]

This method is the same as `find_many()`, except that this method will only return the first result, by way of passing `0` to that method's `start_result_index`, and `end_result_index` parameters.

Parameters

- **query_dict** –
- **projection** –
- **revision** –
- **node_key** –
- **hash** –
- **time_axis_shift** –

Returns

history(*node_key: int*, *subtree: bool = True*, *revision: Optional[Union[int, datetime.datetime]] = None*) → Union[List[*pysirix.types.SubtreeRevision*], List[*pysirix.types.Revision*]]

This method returns the history of a node in the resource.

Parameters

- **node_key** – the root of the subtree whose history should be returned. Defaults to document root.
- **subtree** – whether to account for changes in the subtree of the given node. Defaults to `True`.
- **revision** – the revision in which the node with the given `node_key` exists (if it does not exist currently). Defaults to the latest revision. May be an integer or a `datetime` instance

Returns If `subtree` is `True`, a list of *`pysirix.types.SubtreeRevision`*. Else, a list of `pysirix.types.RevisionType`.

history_embed(*node_key: int*, *revision: Optional[Union[int, datetime.datetime]] = None*) → List[*pysirix.types.QueryResult*]

insert_many(*insert_list: Union[str, List[Dict]]*) → Union[str, Awaitable[str]]

Inserts a list of records into the store. New records are added at the the tail of the store.

Parameters **insert_list** – either a JSON string of `list` of `dicts`, or a `list` that can be converted to JSON

Returns a `str` "{rest: []}" or an `Awaitable[str]` resolving to this string.

insert_one(*insert_dict: Union[str, Dict]*) → Union[str, Awaitable[str]]

Inserts a single record into the store. New records are added at the tail of the store.

Parameters **insert_dict** – either a JSON string of a `dict`, or a `dict` that can be converted to JSON.

Returns an emtpy `str` or an empty `Awaitable[str]`.

name

resource_history() → List[*pysirix.types.Commit*]

This method returns the entire history of a resource.

Returns a list of `Commit`.

update_by_key(*node_key: int*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **node_key** – the nodeKey of the record to update
- **update_dict** – a dict of keys and matching values to replace in the given record
- **upsert** – whether to insert if the field does not already exist

Returns

update_many(*query_dict: Dict*, *update_dict: Dict[str, Optional[Union[List, Dict, str, int]]]*, *upsert: bool = True*) → Union[str, Awaitable[str]]

Parameters

- **query_dict** – a `dict` of field names and their values to match against
- **update_dict** – a dict of keys and matching values to replace in the selected record
- **upsert** – whether to insert if the field does not already exist

Returns

pysirix.json_store.**parse_revision**(*revision: Union[int, datetime.datetime]*, *params: Dict*) → None

pysirix.json_store.**stringify**(*v: Union[None, int, str, Dict, List]*)

1.7 pysirix.types module

class pysirix.types.**Commit**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

author: str

commitMessage: str

revision: int

revisionTimestamp: str

class pysirix.types.**DeleteDiff**(**args*, ***kwargs*)

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

depth: int

deweyID: str

nodeKey: int

`class pysirix.types.`**`InsertDiff`**`(*args, **kwargs)`

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

`data:` *`pysirix.info.DataType`*

`depth:` **`int`**

`deweyID:` **`str`**

`insertPosition:` *`pysirix.info.InsertPosition`*

`insertPositionNodeKey:` **`int`**

`nodeKey:` **`int`**

`type:` **`str`**

`class pysirix.types.`**`MetaNode`**`(*args, **kwargs)`

Bases: `dict`

`key` is provided only if `type` is *`pysirix.info.NodeType`* `OBJECT_KEY`.

`value` is of type `List[MetaNode]` if `metadata.type` is `OBJECT` or `ARRAY`, however, if `metadata.childCount` is 0, then `value` is an emtpy `dict`, or an empty `list`, depending on whether `metadata.type` is `OBJECT` or `ARRAY`.

`value` is of type *`MetaNode`* if `metadata.type` is `OBJECT_KEY`.

`value` is a `str` if `metadata.type` is `OBJECT_STRING_VALUE` or `STRING_VALUE`.

`value` is an `int` or `float` if `metadata.type` == `OBJECT_NUMBER_VALUE` or `NUMBER_VALUE`.

`value` is a `bool` if `metadata.type` is `OBJECT_BOOLEAN_VALUE` or `BOOLEAN_VALUE`.

`value` is `None` if `metadata.type` is `OBJECT_NULL_VALUE` or `NULL_VALUE`.

`key:` **`str`**

`metadata:` *`pysirix.types.Metadata`*

`value:` **`Optional[Union[List[Iterable[`***`pysirix.types.MetaNode`***`]], Iterable[`***`pysirix.types.MetaNode`***`], str, int, float, bool]]`**

`class pysirix.types.`**`Metadata`**`(*args, **kwargs)`

Bases: `dict`

`descendantCount` and `childCount` are provided only where `type` is *`pysirix.info.NodeType`* OBJECT or ARRAY.

`childCount:` **`int`**

`descendantCount:` **`int`**

`hash:` **`int`**

`nodeKey:` **`int`**

`type:` *`pysirix.info.NodeType`*

`class pysirix.types.`**`QueryResult`**`(*args, **kwargs)`

Bases: `dict`

This type is available only in python 3.8+. Otherwise, defaults to `dict`.

`revision:` **`Union[Dict, List]`**

`revisionNumber:` **`int`**

revisionTimestamp: str

class pysirix.types.**ReplaceDiff**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

data: str

nodeKey: int

type: *pysirix.info.DataType*

class pysirix.types.**Revision**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

revision: Optional[Union[List, Dict, str, int, float]]

revisionNumber: int

revisionTimestamp: str

class pysirix.types.**SubtreeRevision**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

revisionNumber: int

revisionTimestamp: str

class pysirix.types.**UpdateDiff**(**args*, ***kwargs*)

Bases: dict

This type is available only in python 3.8+. Otherwise, defaults to dict.

nodeKey: int

type: *pysirix.info.DataType*

value: Optional[Union[str, int, float, bool]]

1.8 pysirix.auth module

class pysirix.auth.**Auth**(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*, *asynchronous: bool*)

This class handles authentication for server access.

__init__(*username: str*, *password: str*, *client: Union[httpx.Client, httpx.AsyncClient]*, *asynchronous: bool*)

Parameters

- **username** – the username for this application.
- **password** – the password for this application.
- **client** – the httpx.Client or httpx.AsyncClient instance used for connecting to the server.
- **asynchronous** – whether or not this application is asynchronous.

async _async_authenticate()
Initial authentication, for asynchronous applications.

async _async_handle_data(*resp*)
Parse token data, and create an asynchronous task to refresh the access token again before it expires.

Parameters **resp** – the `httpx.Response` object.

async _async_refresh()
Refresh the access token, using the refresh token. For asynchronous applications.

_authenticate()
Initial authentication, for synchronous, threaded applications.

_handle_data(*resp*)
Parse token data, and set a `threading.Timer` to refresh the access token again before it expires.

Parameters **resp** – the `httpx.Response` object.

_refresh()
Refresh the access token, using the refresh token. For synchronous, threaded applications.

async _sleep_then_refresh()
Helper function for *_async_handle_data()*. This method sleeps, then calls *_async_refresh()* 10 seconds before the access token is set to expire.

authenticate() → Union[None, Awaitable[None]]
Initial authentication for server access, using username and password. Access tokens are renewed in the background.

dispose()
Remove the authentication timer.

1.9 pysirix.sync_client module

class pysirix.sync_client.**SyncClient**(*client: httpx.Client*)

__init__(*client: httpx.Client*)
The methods of this class call all SirixDB endpoints, with minimal handling. This class is used for synchronous calls, the `AsyncClient` handles asynchronous calls.

Parameters **client** – an instance of `httpx.Client`.

create_database(*name: str*, *db_type:* pysirix.constants.DBType) → None
Call the `/{database}` endpoint with a PUT request

Parameters

- **name** – name of the database to create.
- **db_type** – type of the database to create.

Raises *pysirix.SirixServerError*.

create_resource(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *data: Union[str, bytes, Iterator[bytes]]*, *hash_type: str = 'ROLLING'*) → str
Call the `/{database}/{resource}` endpoint with a PUT request.

Parameters

- **db_name** – the name of the database.

- **db_type** – the type of the database.
- **name** – the name of the resource.
- **data** – the data to initialize the database with.

Returns a `str` of `data`.

Raises *pysirix.SirixServerError*.

delete_all() → None

Call the `/` endpoint with DELETE request. Deletes all databases and their resources. Be careful!

Raises *pysirix.SirixServerError*.

delete_database(*name: str*) → None

call the `/{database}` endpoint with a DELETE request.

Parameters **name** – the name of the database to delete.

Raises *pysirix.SirixServerError*.

diff(*db_name: str*, *name: str*, *params: Dict[str, str]*) → List[Dict[str, Union[*pysirix.types.InsertDiff*, *pysirix.types.ReplaceDiff*, *pysirix.types.UpdateDiff*, int]]]

Call the `/{database}/{resource}/diff` endpoint with a GET request.

Parameters

- **db_name** – the name of the database.
- **name** – the name of the resource.
- **params** – the parameters required for this request.

Returns

get_database_info(*name: str*) → Dict

Call the `/{database}` endpoint with a GET request.

Parameters **name** – name of the database.

Returns a `dict` with a `resources` field containing a `list` of resources.

Raises *pysirix.SirixServerError*.

get_etag(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *params: Dict[str, Union[str, int]]*) → str

Call the `/{database}/{resource}` endpoint with a HEAD request.

Parameters

- **db_name** – the name of the database.
- **db_type** – the type of the database.
- **name** – the name of the resource.
- **params** – the query parameters to call the endpoint with.

Returns the ETag of the node queried.

Raises *pysirix.SirixServerError*.

global_info(*resources: bool = True*) → List[Dict]

Call the `/` endpoint with a GET request. If `resources` is `True`, the endpoint is called with the query `withResources=true`

Parameters **resources** – whether to query resources as well

Returns a `list` of `dict`s, where each `dict` has a `name` field, a `type` field, and (if `resources` is `True`) a `resources` field (containing a `list` of names).

Raises *`pysirix.SirixServerError`*.

`history`(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*) → List[*pysirix.types.Commit*]

Call the `/{database}/{resource}/history` endpoint with a GET request.

Parameters

- **`db_name`** – the name of the database.
- **`db_type`** – the type of the database.
- **`name`** – the name of the resource.

Returns a `list` of `dict` containing the history of the resource.

Raises *`pysirix.SirixServerError`*.

`post_query`(*query: Dict[str, Union[str, int]]*) → str

Call the `/` endpoint with a POST request.

Parameters **`query`** – the body of the request.

Returns the query result as a `str`.

Raises *`pysirix.SirixServerError`*.

`read_resource`(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *params: Dict[str, Union[str, int]]*) → Union[Dict, List, xml.etree.ElementTree.Element]

Call the `/{database}/{resource}` endpoint with a GET request.

Parameters

- **`db_name`** – the name of the database.
- **`db_type`** – the type of the database.
- **`name`** – the name of the resource.
- **`params`** – query parameters to call the endpoint with.

Returns either a `dict` or a `xml.etree.ElementTree.Element`, depending on the database type.

Raises *`pysirix.SirixServerError`*.

`resource_delete`(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *node_id: Optional[int]*, *etag: Optional[str]*) → None

Call the `/{database}/{resource}` endpoint with a DELETE request.

Parameters

- **`db_name`** – the name of the database.
- **`db_type`** – the type of the database.
- **`name`** – the name of the resource.
- **`node_id`** – the nodeKey of the node to delete. `None` to delete the entire resource.
- **`etag`** – the etag of the node to delete.

Raises *`pysirix.SirixServerError`*.

`resource_exists`(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*) → bool

Call the `/{database}/{resource}` endpoint with a HEAD request.

Parameters

- **db_name** – the name of the database.
- **db_type** – the type of the database.
- **name** – the name of the resource.

Returns a `bool` indicating the existence (or lack thereof) of the resource.

Raises *pysirix.SirixServerError*.

update(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *node_id: int*, *data: Union[str, bytes, Iterator[bytes]]*, *insert:* pysirix.constants.Insert, *etag: Optional[str]*) → str

Call the `/{database}/{resource}` endpoint with a POST request.

Parameters

- **db_name** – the name of the database.
- **db_type** – the type of the database.
- **name** – the name of the resource.
- **node_id** – the nodeKey of the node in relation to which the update is performed.
- **data** – the data used in the update operation.
- **insert** – the position of the update in relation to the node referenced by node_id.
- **etag** – the ETag of the node referenced by node_id.

Returns the resource as a `str`.

Raises *pysirix.SirixServerError*.

1.10 pysirix.async_client module

class pysirix.async_client.**AsyncClient**(*client: httpx.AsyncClient*)

__init__(*client: httpx.AsyncClient*)

The methods of this class call all SirixDB endpoints, with minimal handling. This class is used for asynchronous calls, the `SyncClient` handles synchronous calls. All methods of this class are identical to those of `SyncClient` (with the distinction that the methods of this class are asynchronous), and are not documented here again.

Parameters **client** – an instance of `httpx.AsyncClient`.

async create_database(*name: str*, *db_type:* pysirix.constants.DBType) → None

async create_resource(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *data: Union[str, bytes, Iterator[bytes], AsyncIterator[bytes]]*, *hash_type: str = 'ROLLING'*) → str

async delete_all() → None

async delete_database(*name: str*) → None

async diff(*db_name: str*, *name: str*, *params: Dict[str, str]*) → List[Dict[str, Union[*pysirix.types.InsertDiff*, *pysirix.types.ReplaceDiff*, *pysirix.types.UpdateDiff*, int]]]

async get_database_info(*name: str*) → Dict

async get_etag(*db_name: str*, *db_type:* pysirix.constants.DBType, *name: str*, *params: Dict[str, Union[str, int]]*) → str

async global_info(*resources=True*) → List[Dict]

async history(*db_name: str*, *db_type: pysirix.constants.DBType*, *name: str*) → List[*pysirix.types.Commit*]

async post_query(*query: Dict[str, Union[str, int]]*)

async read_resource(*db_name: str*, *db_type: pysirix.constants.DBType*, *name: str*, *params: Dict[str, Union[str, int]]*) → Union[Dict, List, xml.etree.ElementTree.Element]

async resource_delete(*db_name: str*, *db_type: pysirix.constants.DBType*, *name: str*, *node_id: Optional[int]*, *etag: Optional[str]*) → None

async resource_exists(*db_name: str*, *db_type: pysirix.constants.DBType*, *name: str*) → bool

async update(*db_name: str*, *db_type: pysirix.constants.DBType*, *name: str*, *node_id: int*, *data: Union[str, bytes, Iterator[bytes], AsyncIterator[bytes]]*, *insert: pysirix.constants.Insert*, *etag: Optional[str]*) → str

1.11 pysirix.errors module

exception pysirix.errors.**SirixServerError**(*message: str*, ***, *request: Request*, *response: Response*)

pysirix.errors.**include_response_text_in_errors**()

1.12 pysirix.info module

class pysirix.info.**DataType**(*value*)

An enumeration.

boolean = 'boolean'

jsonFragment = 'jsonFragment'

null = 'null'

number = 'number'

string = 'string'

class pysirix.info.**InsertPosition**(*value*)

An enumeration.

asFirstChild = 'asFirstChild'

asLeftSibling = 'asLeftSibling'

asRightSibling = 'asRightSibling'

replace = 'replace'

class pysirix.info.**NodeType**(*value*)

An enumeration.

ARRAY = 'ARRAY'

BOOLEAN_VALUE = 'BOOLEAN_VALUE'

NULL_VALUE = 'NULL_VALUE'

NUMBER_VALUE = 'NUMBER_VALUE'

OBJECT = 'OBJECT'

`OBJECT_BOOLEAN_VALUE = 'OBJECT_BOOLEAN_VALUE'`

`OBJECT_KEY = 'OBJECT_KEY'`

`OBJECT_NULL_VALUE = 'OBJECT_NULL_VALUE'`

`OBJECT_NUMBER_VALUE = 'OBJECT_NUMBER_VALUE'`

`OBJECT_STRING_VALUE = 'OBJECT_STRING_VALUE'`

`STRING_VALUE = 'STRING_VALUE'`

1.13 pysirix.constants module

class `pysirix.constants.`**`DBType`**(*value*)

Bases: `enum.Enum`

This Enum class defines the possible database (and resource) types

`JSON = 'application/json'`

`XML = 'application/xml'`

class `pysirix.constants.`**`Insert`**(*value*)

Bases: `enum.Enum`

This Enum class defines the possible options for a resource update

`CHILD = 'asFirstChild'`

`LEFT = 'asLeftSibling'`

`REPLACE = 'replace'`

`RIGHT = 'asRightSibling'`

class `pysirix.constants.`**`MetadataType`**(*value*)

Bases: `enum.Enum`

This class defines the scope of the metadata to return using the `readWithMetadata()` method.

`ALL = True`

`KEY = 'nodeKey'`

`KEYAndCHILD = 'nodeKeyAndChildCount'`

class `pysirix.constants.`**`TimeAxisShift`**(*value*)

Bases: `enum.Enum`

An enumeration.

`latest = 1`

`none = 0`

`oldest = -1`

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX