

---

**PySirix**  
*Release 0.5.1*

**Moshe Uminer**

**Aug 17, 2023**



## CONTENTS:

<b>1</b>	<b>pysirix package</b>	<b>1</b>
1.1	Module contents . . . . .	1
1.2	Submodules . . . . .	10
1.3	pysirix.sirix module . . . . .	10
1.4	pysirix.database module . . . . .	11
1.5	pysirix.resource module . . . . .	12
1.6	pysirix.json_store module . . . . .	15
1.7	pysirix.types module . . . . .	24
1.8	pysirix.auth module . . . . .	26
1.9	pysirix.sync_client module . . . . .	27
1.10	pysirix.async_client module . . . . .	30
1.11	pysirix.errors module . . . . .	31
1.12	pysirix.info module . . . . .	31
1.13	pysirix.constants module . . . . .	32
<b>2</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>
	<b>Index</b>	<b>37</b>



## 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 on 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 empty 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 on 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.type.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 empty 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 empty 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 empty 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 empty 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 dicts, 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
```

## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### p

- pysirix, 1
- pysirix.async\_client, 30
- pysirix.auth, 26
- pysirix.constants, 32
- pysirix.database, 11
- pysirix.errors, 31
- pysirix.info, 31
- pysirix.json\_store, 15
- pysirix.resource, 12
- pysirix.sirix, 10
- pysirix.sync\_client, 27
- pysirix.types, 24





## Symbols

- `__init__()` (*pysirix.Database* method), 1
  - `__init__()` (*pysirix.Resource* method), 6
  - `__init__()` (*pysirix.Sirix* method), 8
  - `__init__()` (*pysirix.async\_client.AsyncClient* method), 30
  - `__init__()` (*pysirix.auth.Auth* method), 26
  - `__init__()` (*pysirix.database.Database* method), 11
  - `__init__()` (*pysirix.json\_store.JsonStoreAsync* method), 15
  - `__init__()` (*pysirix.json\_store.JsonStoreBase* method), 18
  - `__init__()` (*pysirix.json\_store.JsonStoreSync* method), 21
  - `__init__()` (*pysirix.resource.Resource* method), 12
  - `__init__()` (*pysirix.sirix.Sirix* method), 10
  - `__init__()` (*pysirix.sync\_client.SyncClient* method), 27
  - `_abc_impl` (*pysirix.json\_store.JsonStoreAsync* attribute), 15
  - `_abc_impl` (*pysirix.json\_store.JsonStoreBase* attribute), 18
  - `_abc_impl` (*pysirix.json\_store.JsonStoreSync* attribute), 21
  - `_async_authenticate()` (*pysirix.auth.Auth* method), 26
  - `_async_handle_data()` (*pysirix.auth.Auth* method), 27
  - `_async_refresh()` (*pysirix.auth.Auth* method), 27
  - `_auth` (*pysirix.json\_store.JsonStoreAsync* attribute), 15
  - `_auth` (*pysirix.json\_store.JsonStoreBase* attribute), 18
  - `_auth` (*pysirix.json\_store.JsonStoreSync* attribute), 21
  - `_authenticate()` (*pysirix.auth.Auth* method), 27
  - `_build_read_params()` (*pysirix.resource.Resource* static method), 12
  - `_client` (*pysirix.json\_store.JsonStoreAsync* attribute), 15
  - `_client` (*pysirix.json\_store.JsonStoreBase* attribute), 18
  - `_client` (*pysirix.json\_store.JsonStoreSync* attribute), 21
  - `_handle_data()` (*pysirix.auth.Auth* method), 27
  - `_prepare_find_all()` (*pysirix.json\_store.JsonStoreAsync* method), 15
  - `_prepare_find_all()` (*pysirix.json\_store.JsonStoreBase* method), 18
  - `_prepare_find_all()` (*pysirix.json\_store.JsonStoreSync* method), 21
  - `_refresh()` (*pysirix.auth.Auth* method), 27
  - `_sleep_then_refresh()` (*pysirix.auth.Auth* method), 27
- ## A
- ALL (*pysirix.constants.MetadataType* attribute), 32
  - ARRAY (*pysirix.info.NodeType* attribute), 31
  - asFirstChild (*pysirix.info.InsertPosition* attribute), 31
  - asLeftSibling (*pysirix.info.InsertPosition* attribute), 31
  - asRightSibling (*pysirix.info.InsertPosition* attribute), 31
  - AsyncClient (*class in pysirix.async\_client*), 30
  - Auth (*class in pysirix.auth*), 26
  - authenticate() (*pysirix.auth.Auth* method), 27
  - authenticate() (*pysirix.Sirix* method), 9
  - authenticate() (*pysirix.sirix.Sirix* method), 10
  - author (*pysirix.Commit* attribute), 1
  - author (*pysirix.types.Commit* attribute), 24
- ## B
- boolean (*pysirix.info.DataType* attribute), 31
  - BOOLEAN\_VALUE (*pysirix.info.NodeType* attribute), 31
- ## C
- CHILD (*pysirix.constants.Insert* attribute), 32
  - CHILD (*pysirix.Insert* attribute), 2
  - childCount (*pysirix.Metadata* attribute), 6
  - childCount (*pysirix.types.Metadata* attribute), 25
  - Commit (*class in pysirix*), 1
  - Commit (*class in pysirix.types*), 24
  - commitMessage (*pysirix.Commit* attribute), 1
  - commitMessage (*pysirix.types.Commit* attribute), 24
  - create() (*pysirix.Database* method), 1
  - create() (*pysirix.database.Database* method), 11
  - create() (*pysirix.json\_store.JsonStoreAsync* method), 15
  - create() (*pysirix.json\_store.JsonStoreBase* method), 18

create() (*pysirix.json\_store.JsonStoreSync method*), 21  
 create() (*pysirix.Resource method*), 7  
 create() (*pysirix.resource.Resource method*), 13  
 create\_database() (*pysirix.async\_client.AsyncClient method*), 30  
 create\_database() (*pysirix.sync\_client.SyncClient method*), 27  
 create\_resource() (*pysirix.async\_client.AsyncClient method*), 30  
 create\_resource() (*pysirix.sync\_client.SyncClient method*), 27

## D

data (*pysirix.InsertDiff attribute*), 2  
 data (*pysirix.ReplaceDiff attribute*), 6  
 data (*pysirix.types.InsertDiff attribute*), 25  
 data (*pysirix.types.ReplaceDiff attribute*), 26  
 Database (*class in pysirix*), 1  
 Database (*class in pysirix.database*), 11  
 database() (*pysirix.Sirix method*), 9  
 database() (*pysirix.sirix.Sirix method*), 10  
 DataType (*class in pysirix.info*), 31  
 db\_name (*pysirix.json\_store.JsonStoreAsync attribute*), 15  
 db\_name (*pysirix.json\_store.JsonStoreBase attribute*), 18  
 db\_name (*pysirix.json\_store.JsonStoreSync attribute*), 21  
 db\_name (*pysirix.JsonStoreAsync attribute*), 3  
 db\_name (*pysirix.JsonStoreSync attribute*), 4  
 db\_type (*pysirix.json\_store.JsonStoreAsync attribute*), 15  
 db\_type (*pysirix.json\_store.JsonStoreBase attribute*), 18  
 db\_type (*pysirix.json\_store.JsonStoreSync attribute*), 21  
 db\_type (*pysirix.JsonStoreAsync attribute*), 3  
 db\_type (*pysirix.JsonStoreSync attribute*), 4  
 DBType (*class in pysirix*), 1  
 DBType (*class in pysirix.constants*), 32  
 delete() (*pysirix.Database method*), 1  
 delete() (*pysirix.database.Database method*), 11  
 delete() (*pysirix.Resource method*), 7  
 delete() (*pysirix.resource.Resource method*), 13  
 delete\_all() (*pysirix.async\_client.AsyncClient method*), 30  
 delete\_all() (*pysirix.Sirix method*), 9  
 delete\_all() (*pysirix.sirix.Sirix method*), 11  
 delete\_all() (*pysirix.sync\_client.SyncClient method*), 28  
 delete\_database() (*pysirix.async\_client.AsyncClient method*), 30  
 delete\_database() (*pysirix.sync\_client.SyncClient method*), 28  
 delete\_field() (*pysirix.json\_store.JsonStoreAsync method*), 15  
 delete\_field() (*pysirix.json\_store.JsonStoreBase method*), 18

delete\_field() (*pysirix.json\_store.JsonStoreSync method*), 21  
 delete\_fields\_by\_key() (*pysirix.json\_store.JsonStoreAsync method*), 15  
 delete\_fields\_by\_key() (*pysirix.json\_store.JsonStoreBase method*), 18  
 delete\_fields\_by\_key() (*pysirix.json\_store.JsonStoreSync method*), 22  
 delete\_records() (*pysirix.json\_store.JsonStoreAsync method*), 15  
 delete\_records() (*pysirix.json\_store.JsonStoreBase method*), 19  
 delete\_records() (*pysirix.json\_store.JsonStoreSync method*), 22  
 DeleteDiff (*class in pysirix*), 2  
 DeleteDiff (*class in pysirix.types*), 24  
 depth (*pysirix.DeleteDiff attribute*), 2  
 depth (*pysirix.InsertDiff attribute*), 2  
 depth (*pysirix.types.DeleteDiff attribute*), 24  
 depth (*pysirix.types.InsertDiff attribute*), 25  
 descendantCount (*pysirix.Metadata attribute*), 6  
 descendantCount (*pysirix.types.Metadata attribute*), 25  
 deweyID (*pysirix.DeleteDiff attribute*), 2  
 deweyID (*pysirix.InsertDiff attribute*), 2  
 deweyID (*pysirix.types.DeleteDiff attribute*), 24  
 deweyID (*pysirix.types.InsertDiff attribute*), 25  
 diff() (*pysirix.async\_client.AsyncClient method*), 30  
 diff() (*pysirix.Resource method*), 7  
 diff() (*pysirix.resource.Resource method*), 13  
 diff() (*pysirix.sync\_client.SyncClient method*), 28  
 dispose() (*pysirix.auth.Auth method*), 27  
 dispose() (*pysirix.Sirix method*), 9  
 dispose() (*pysirix.sirix.Sirix method*), 11

## E

exists() (*pysirix.json\_store.JsonStoreAsync method*), 16  
 exists() (*pysirix.json\_store.JsonStoreBase method*), 19  
 exists() (*pysirix.json\_store.JsonStoreSync method*), 22  
 exists() (*pysirix.Resource method*), 7  
 exists() (*pysirix.resource.Resource method*), 13

## F

find\_all() (*pysirix.json\_store.JsonStoreAsync method*), 16  
 find\_all() (*pysirix.json\_store.JsonStoreBase method*), 19  
 find\_all() (*pysirix.json\_store.JsonStoreSync method*), 22  
 find\_all() (*pysirix.JsonStoreAsync method*), 3  
 find\_all() (*pysirix.JsonStoreSync method*), 4

- find\_by\_key() (*pysirix.json\_store.JsonStoreAsync method*), 16  
 find\_by\_key() (*pysirix.json\_store.JsonStoreBase method*), 19  
 find\_by\_key() (*pysirix.json\_store.JsonStoreSync method*), 22  
 find\_by\_key() (*pysirix.JsonStoreAsync method*), 3  
 find\_by\_key() (*pysirix.JsonStoreSync method*), 5  
 find\_one() (*pysirix.json\_store.JsonStoreAsync method*), 16  
 find\_one() (*pysirix.json\_store.JsonStoreBase method*), 19  
 find\_one() (*pysirix.json\_store.JsonStoreSync method*), 23
- ## G
- get\_database\_info() (*pysirix.async\_client.AsyncClient method*), 30  
 get\_database\_info() (*pysirix.Database method*), 2  
 get\_database\_info() (*pysirix.database.Database method*), 12  
 get\_database\_info() (*pysirix.sync\_client.SyncClient method*), 28  
 get\_etag() (*pysirix.async\_client.AsyncClient method*), 30  
 get\_etag() (*pysirix.Resource method*), 7  
 get\_etag() (*pysirix.resource.Resource method*), 13  
 get\_etag() (*pysirix.sync\_client.SyncClient method*), 28  
 get\_info() (*pysirix.Sirix method*), 9  
 get\_info() (*pysirix.sirix.Sirix method*), 11  
 global\_info() (*pysirix.async\_client.AsyncClient method*), 31  
 global\_info() (*pysirix.sync\_client.SyncClient method*), 28
- ## H
- hash (*pysirix.Metadata attribute*), 6  
 hash (*pysirix.types.Metadata attribute*), 25  
 history() (*pysirix.async\_client.AsyncClient method*), 31  
 history() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 history() (*pysirix.json\_store.JsonStoreBase method*), 20  
 history() (*pysirix.json\_store.JsonStoreSync method*), 23  
 history() (*pysirix.JsonStoreAsync method*), 3  
 history() (*pysirix.JsonStoreSync method*), 5  
 history() (*pysirix.Resource method*), 7  
 history() (*pysirix.resource.Resource method*), 13  
 history() (*pysirix.sync\_client.SyncClient method*), 29  
 history\_embed() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 history\_embed() (*pysirix.json\_store.JsonStoreBase method*), 20  
 history\_embed() (*pysirix.json\_store.JsonStoreSync method*), 23  
 history\_embed() (*pysirix.JsonStoreAsync method*), 4  
 history\_embed() (*pysirix.JsonStoreSync method*), 5
- ## I
- include\_response\_text\_in\_errors() (*in module pysirix.errors*), 31  
 Insert (*class in pysirix*), 2  
 Insert (*class in pysirix.constants*), 32  
 insert\_many() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 insert\_many() (*pysirix.json\_store.JsonStoreBase method*), 20  
 insert\_many() (*pysirix.json\_store.JsonStoreSync method*), 23  
 insert\_one() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 insert\_one() (*pysirix.json\_store.JsonStoreBase method*), 20  
 insert\_one() (*pysirix.json\_store.JsonStoreSync method*), 23  
 InsertDiff (*class in pysirix*), 2  
 InsertDiff (*class in pysirix.types*), 24  
 InsertPosition (*class in pysirix.info*), 31  
 insertPosition (*pysirix.InsertDiff attribute*), 2  
 insertPosition (*pysirix.types.InsertDiff attribute*), 25  
 insertPositionNodeKey (*pysirix.InsertDiff attribute*), 2  
 insertPositionNodeKey (*pysirix.types.InsertDiff attribute*), 25
- ## J
- JSON (*pysirix.constants.DBType attribute*), 32  
 JSON (*pysirix.DBType attribute*), 1  
 json\_store() (*pysirix.Database method*), 2  
 json\_store() (*pysirix.database.Database method*), 12  
 jsonFragment (*pysirix.info.DataType attribute*), 31  
 JsonStoreAsync (*class in pysirix*), 3  
 JsonStoreAsync (*class in pysirix.json\_store*), 15  
 JsonStoreBase (*class in pysirix.json\_store*), 18  
 JsonStoreSync (*class in pysirix*), 4  
 JsonStoreSync (*class in pysirix.json\_store*), 21
- ## K
- KEY (*pysirix.constants.MetadataType attribute*), 32  
 key (*pysirix.MetaNode attribute*), 6  
 key (*pysirix.types.MetaNode attribute*), 25  
 KEYAndCHILD (*pysirix.constants.MetadataType attribute*), 32

## L

latest (*pysirix.constants.TimeAxisShift* attribute), 32  
 latest (*pysirix.TimeAxisShift* attribute), 9  
 LEFT (*pysirix.constants.Insert* attribute), 32  
 LEFT (*pysirix.Insert* attribute), 2

## M

Metadata (*class in pysirix*), 6  
 Metadata (*class in pysirix.types*), 25  
 metadata (*pysirix.MetaNode* attribute), 6  
 metadata (*pysirix.types.MetaNode* attribute), 25  
 MetadataType (*class in pysirix.constants*), 32  
 MetaNode (*class in pysirix*), 5  
 MetaNode (*class in pysirix.types*), 25  
 module  
   pysirix, 1  
   pysirix.async\_client, 30  
   pysirix.auth, 26  
   pysirix.constants, 32  
   pysirix.database, 11  
   pysirix.errors, 31  
   pysirix.info, 31  
   pysirix.json\_store, 15  
   pysirix.resource, 12  
   pysirix.sirix, 10  
   pysirix.sync\_client, 27  
   pysirix.types, 24

## N

name (*pysirix.json\_store.JsonStoreAsync* attribute), 17  
 name (*pysirix.json\_store.JsonStoreBase* attribute), 20  
 name (*pysirix.json\_store.JsonStoreSync* attribute), 23  
 name (*pysirix.JsonStoreAsync* attribute), 4  
 name (*pysirix.JsonStoreSync* attribute), 5  
 nodeKey (*pysirix.DeleteDiff* attribute), 2  
 nodeKey (*pysirix.InsertDiff* attribute), 2  
 nodeKey (*pysirix.Metadata* attribute), 6  
 nodeKey (*pysirix.ReplaceDiff* attribute), 6  
 nodeKey (*pysirix.types.DeleteDiff* attribute), 24  
 nodeKey (*pysirix.types.InsertDiff* attribute), 25  
 nodeKey (*pysirix.types.Metadata* attribute), 25  
 nodeKey (*pysirix.types.ReplaceDiff* attribute), 26  
 nodeKey (*pysirix.types.UpdateDiff* attribute), 26  
 nodeKey (*pysirix.UpdateDiff* attribute), 10  
 NodeType (*class in pysirix.info*), 31  
 none (*pysirix.constants.TimeAxisShift* attribute), 32  
 none (*pysirix.TimeAxisShift* attribute), 9  
 null (*pysirix.info.DataType* attribute), 31  
 NULL\_VALUE (*pysirix.info.NodeType* attribute), 31  
 number (*pysirix.info.DataType* attribute), 31  
 NUMBER\_VALUE (*pysirix.info.NodeType* attribute), 31

## O

OBJECT (*pysirix.info.NodeType* attribute), 31

OBJECT\_BOOLEAN\_VALUE (*pysirix.info.NodeType* attribute), 32  
 OBJECT\_KEY (*pysirix.info.NodeType* attribute), 32  
 OBJECT\_NULL\_VALUE (*pysirix.info.NodeType* attribute), 32  
 OBJECT\_NUMBER\_VALUE (*pysirix.info.NodeType* attribute), 32  
 OBJECT\_STRING\_VALUE (*pysirix.info.NodeType* attribute), 32  
 oldest (*pysirix.constants.TimeAxisShift* attribute), 32  
 oldest (*pysirix.TimeAxisShift* attribute), 9

## P

parse\_revision() (*in module pysirix.json\_store*), 24  
 post\_query() (*pysirix.async\_client.AsyncClient* method), 31  
 post\_query() (*pysirix.sync\_client.SyncClient* method), 29  
 pysirix  
   module, 1  
 pysirix.async\_client  
   module, 30  
 pysirix.auth  
   module, 26  
 pysirix.constants  
   module, 32  
 pysirix.database  
   module, 11  
 pysirix.errors  
   module, 31  
 pysirix.info  
   module, 31  
 pysirix.json\_store  
   module, 15  
 pysirix.resource  
   module, 12  
 pysirix.sirix  
   module, 10  
 pysirix.sync\_client  
   module, 27  
 pysirix.types  
   module, 24

## Q

query() (*pysirix.Resource* method), 7  
 query() (*pysirix.resource.Resource* method), 13  
 query() (*pysirix.Sirix* method), 9  
 query() (*pysirix.sirix.Sirix* method), 11  
 QueryResult (*class in pysirix*), 6  
 QueryResult (*class in pysirix.types*), 25

## R

read() (*pysirix.Resource* method), 7  
 read() (*pysirix.resource.Resource* method), 13

- read\_resource() (*pysirix.async\_client.AsyncClient method*), 31  
 read\_resource() (*pysirix.sync\_client.SyncClient method*), 29  
 read\_with\_metadata() (*pysirix.Resource method*), 8  
 read\_with\_metadata() (*pysirix.resource.Resource method*), 14  
 REPLACE (*pysirix.constants.Insert attribute*), 32  
 replace (*pysirix.info.InsertPosition attribute*), 31  
 REPLACE (*pysirix.Insert attribute*), 2  
 ReplaceDiff (*class in pysirix*), 6  
 ReplaceDiff (*class in pysirix.types*), 26  
 Resource (*class in pysirix*), 6  
 Resource (*class in pysirix.resource*), 12  
 resource() (*pysirix.Database method*), 2  
 resource() (*pysirix.database.Database method*), 12  
 resource\_delete() (*pysirix.async\_client.AsyncClient method*), 31  
 resource\_delete() (*pysirix.sync\_client.SyncClient method*), 29  
 resource\_exists() (*pysirix.async\_client.AsyncClient method*), 31  
 resource\_exists() (*pysirix.sync\_client.SyncClient method*), 29  
 resource\_history() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 resource\_history() (*pysirix.json\_store.JsonStoreBase method*), 20  
 resource\_history() (*pysirix.json\_store.JsonStoreSync method*), 24  
 resource\_history() (*pysirix.JsonStoreAsync method*), 4  
 resource\_history() (*pysirix.JsonStoreSync method*), 5  
 Revision (*class in pysirix*), 8  
 Revision (*class in pysirix.types*), 26  
 revision (*pysirix.Commit attribute*), 1  
 revision (*pysirix.QueryResult attribute*), 6  
 revision (*pysirix.Revision attribute*), 8  
 revision (*pysirix.types.Commit attribute*), 24  
 revision (*pysirix.types.QueryResult attribute*), 25  
 revision (*pysirix.types.Revision attribute*), 26  
 revisionNumber (*pysirix.QueryResult attribute*), 6  
 revisionNumber (*pysirix.Revision attribute*), 8  
 revisionNumber (*pysirix.types.QueryResult attribute*), 25  
 revisionNumber (*pysirix.types.Revision attribute*), 26  
 revisionNumber (*pysirix.types.SubtreeRevision attribute*), 26  
 revisionTimestamp (*pysirix.Commit attribute*), 1  
 revisionTimestamp (*pysirix.QueryResult attribute*), 6  
 revisionTimestamp (*pysirix.Revision attribute*), 8  
 revisionTimestamp (*pysirix.types.Commit attribute*), 24  
 revisionTimestamp (*pysirix.types.QueryResult attribute*), 26  
 revisionTimestamp (*pysirix.types.Revision attribute*), 26  
 revisionTimestamp (*pysirix.types.SubtreeRevision attribute*), 26  
 RIGHT (*pysirix.constants.Insert attribute*), 32  
 RIGHT (*pysirix.Insert attribute*), 2
- ## S
- Sirix (*class in pysirix*), 8  
 Sirix (*class in pysirix.sirix*), 10  
 sirix\_async() (*in module pysirix*), 10  
 sirix\_sync() (*in module pysirix*), 10  
 SirixServerError, 9, 31  
 string (*pysirix.info.DataType attribute*), 31  
 STRING\_VALUE (*pysirix.info.NodeType attribute*), 32  
 stringify() (*in module pysirix.json\_store*), 24  
 SubtreeRevision (*class in pysirix.types*), 26  
 SyncClient (*class in pysirix.sync\_client*), 27
- ## T
- TimeAxisShift (*class in pysirix*), 9  
 TimeAxisShift (*class in pysirix.constants*), 32  
 type (*pysirix.InsertDiff attribute*), 3  
 type (*pysirix.Metadata attribute*), 6  
 type (*pysirix.ReplaceDiff attribute*), 6  
 type (*pysirix.types.InsertDiff attribute*), 25  
 type (*pysirix.types.Metadata attribute*), 25  
 type (*pysirix.types.ReplaceDiff attribute*), 26  
 type (*pysirix.types.UpdateDiff attribute*), 26  
 type (*pysirix.UpdateDiff attribute*), 10
- ## U
- update() (*pysirix.async\_client.AsyncClient method*), 31  
 update() (*pysirix.Resource method*), 8  
 update() (*pysirix.resource.Resource method*), 14  
 update() (*pysirix.sync\_client.SyncClient method*), 30  
 update\_by\_key() (*pysirix.json\_store.JsonStoreAsync method*), 17  
 update\_by\_key() (*pysirix.json\_store.JsonStoreBase method*), 20  
 update\_by\_key() (*pysirix.json\_store.JsonStoreSync method*), 24  
 update\_many() (*pysirix.json\_store.JsonStoreAsync method*), 18  
 update\_many() (*pysirix.json\_store.JsonStoreBase method*), 21  
 update\_many() (*pysirix.json\_store.JsonStoreSync method*), 24  
 UpdateDiff (*class in pysirix*), 10  
 UpdateDiff (*class in pysirix.types*), 26

## V

- value (*pysirix.MetaNode* attribute), 6
- value (*pysirix.types.MetaNode* attribute), 25
- value (*pysirix.types.UpdateDiff* attribute), 26
- value (*pysirix.UpdateDiff* attribute), 10

## X

- XML (*pysirix.constants.DBType* attribute), 32
- XML (*pysirix.DBType* attribute), 1