python-arango Documentation

Release 4.4.0

Joohwan Oh

Jan 04, 2019

Contents

PYTHON-ARANGO

Welcome to the documentation for **python-arango**, a Python driver for ArangoDB.

- Clean Pythonic interface
- Lightweight
- High ArangoDB REST API coverage

CHAPTER 1

Compatibility

- Python versions 2.7, 3.4, 3.5 and 3.6 are supported
- Python-arango 4.x supports ArangoDB 3.3+ (recommended)
- Python-arango 3.x supports ArangoDB 3.0 ~ 3.2 only
- Python-arango 2.x supports ArangoDB 1.x ~ 2.x only

CHAPTER 2

Installation

To install a stable version from PyPi:

```
~$ pip install python-arango
```

To install the latest version directly from GitHub:

```
~$ pip install -e git+git@github.com:joowani/python-arango.git@master#egg=python-
↪arango
```

You may need to use `sudo` depending on your environment.

CHAPTER 3

Contents

3.1 Getting Started

Here is an example showing how **python-arango** client can be used:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient(protocol='http', host='localhost', port=8529)

# Connect to "_system" database as root user.
# This returns an API wrapper for "_system" database.
sys_db = client.db('_system', username='root', password='passwd')

# Create a new database named "test" if it does not exist.
if not sys_db.has_database('test'):
    sys_db.create_database('test')

# Connect to "test" database as root user.
# This returns an API wrapper for "test" database.
db = client.db('test', username='root', password='passwd')

# Create a new collection named "students" if it does not exist.
# This returns an API wrapper for "students" collection.
if db.has_collection('students'):
    students = db.collection('students')
else:
    students = db.create_collection('students')

# Add a hash index to the collection.
students.add_hash_index(fields=['name'], unique=False)

# Truncate the collection.
students.truncate()
```

(continues on next page)

(continued from previous page)

```
# Insert new documents into the collection.
students.insert({'name': 'jane', 'age': 19})
students.insert({'name': 'josh', 'age': 18})
students.insert({'name': 'jake', 'age': 21})

# Execute an AQL query. This returns a result cursor.
cursor = db.aql.execute('FOR doc IN students RETURN doc')

# Iterate through the cursor to retrieve the documents.
student_names = [document['name'] for document in cursor]
```

3.2 Databases

ArangoDB server can have an arbitrary number of **databases**. Each database has its own set of *collections* and *graphs*. There is a special database named `_system`, which cannot be dropped and provides operations for managing users, permissions and other databases. Most of the operations can only be executed by admin users. See *Users and Permissions* for more information.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient(protocol='http', host='localhost', port=8529)

# Connect to "_system" database as root user.
# This returns an API wrapper for "_system" database.
sys_db = client.db('_system', username='root', password='passwd')

# List all databases.
sys_db.databases()

# Create a new database named "test" if it does not exist.
# Only root user has access to it at time of its creation.
if not sys_db.has_database('test'):
    sys_db.create_database('test')

# Delete the database.
sys_db.delete_database('test')

# Create a new database named "test" along with a new set of users.
# Only "jane", "john", "jake" and root user have access to it.
if not sys_db.has_database('test'):
    sys_db.create_database(
        name='test',
        users=[
            {'username': 'jane', 'password': 'foo', 'active': True},
            {'username': 'john', 'password': 'bar', 'active': True},
            {'username': 'jake', 'password': 'baz', 'active': True},
        ],
    )

# Connect to the new "test" database as user "jane".
```

(continues on next page)

(continued from previous page)

```
db = client.db('test', username='jane', password='foo')

# Make sure that user "jane" has read and write permissions.
sys_db.update_permission(username='jane', permission='rw', database='test')

# Retrieve various database and server information.
db.name
db.username
db.version()
db.status()
db.details()
db.collections()
db.graphs()
db.engine()

# Delete the database. Note that the new users will remain.
sys_db.delete_database('test')
```

See *ArangoClient* and *StandardDatabase* for API specification.

3.3 Collections

A **collection** contains *documents*. It is uniquely identified by its name which must consist only of hyphen, underscore and alphanumeric characters. There are three types of collections in python-arango:

- **Standard Collection:** contains regular documents.
- **Vertex Collection:** contains vertex documents for graphs. See *here* for more details.
- **Edge Collection:** contains edge documents for graphs. See *here* for more details.

Here is an example showing how you can manage standard collections:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List all collections in the database.
db.collections()

# Create a new collection named "students" if it does not exist.
# This returns an API wrapper for "students" collection.
if db.has_collection('students'):
    students = db.collection('students')
else:
    students = db.create_collection('students')

# Retrieve collection properties.
students.name
students.db_name
students.properties()
students.revision()
```

(continues on next page)

(continued from previous page)

```
students.statistics()
students.checksum()
students.count()

# Perform various operations.
students.load()
students.unload()
students.truncate()
students.configure(journal_size=3000000)

# Delete the collection.
db.delete_collection('students')
```

See *StandardDatabase* and *StandardCollection* for API specification.

3.4 Documents

In python-arango, a **document** is a Python dictionary with the following properties:

- Is JSON serializable.
- May be nested to an arbitrary depth.
- May contain lists.
- Contains the `_key` field, which identifies the document uniquely within a specific collection.
- Contains the `_id` field (also called the *handle*), which identifies the document uniquely across all collections within a database. This ID is a combination of the collection name and the document key using the format `{collection}/{key}` (see example below).
- Contains the `_rev` field. ArangoDB supports MVCC (Multiple Version Concurrency Control) and is capable of storing each document in multiple revisions. Latest revision of a document is indicated by this field. The field is populated by ArangoDB and is not required as input unless you want to validate a document against its current revision.

For more information on documents and associated terminologies, refer to ArangoDB manual. Here is an example of a valid document in "students" collection:

```
{
    '_id': 'students/bruce',
    '_key': 'bruce',
    '_rev': '_Wm3dzEi--_',
    'first_name': 'Bruce',
    'last_name': 'Wayne',
    'address': {
        'street' : '1007 Mountain Dr.',
        'city': 'Gotham',
        'state': 'NJ'
    },
    'is_rich': True,
    'friends': ['robin', 'gordon']
}
```

Edge documents (edges) are similar to standard documents but with two additional required fields `_from` and `_to`. Values of these fields must be the handles of "from" and "to" vertex documents linked by the edge document in

question (see *Graphs* for details). Edge documents are contained in *edge collections*. Here is an example of a valid edge document in "friends" edge collection:

```
{
    '_id': 'friends/001',
    '_key': '001',
    '_rev': '_Wm3dyle--_',
    '_from': 'students/john',
    '_to': 'students/jane',
    'closeness': 9.5
}
```

Standard documents are managed via collection API wrapper:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Create some test documents to play around with.
lola = {'_key': 'lola', 'GPA': 3.5, 'first': 'Lola', 'last': 'Martin'}
abby = {'_key': 'abby', 'GPA': 3.2, 'first': 'Abby', 'last': 'Page'}
john = {'_key': 'john', 'GPA': 3.6, 'first': 'John', 'last': 'Kim'}
emma = {'_key': 'emma', 'GPA': 4.0, 'first': 'Emma', 'last': 'Park'}

# Insert a new document. This returns the document metadata.
metadata = students.insert(lola)
assert metadata['_id'] == 'students/lola'
assert metadata['_key'] == 'lola'

# Check if documents exist in the collection in multiple ways.
assert students.has('lola') and 'john' not in students

# Retrieve the total document count in multiple ways.
assert students.count() == len(students) == 1

# Insert multiple documents in bulk.
students.import_bulk([abby, john, emma])

# Retrieve one or more matching documents.
for student in students.find({'first': 'John'}):
    assert student['_key'] == 'john'
    assert student['GPA'] == 3.6
    assert student['last'] == 'Kim'

# Retrieve a document by key.
students.get('john')

# Retrieve a document by ID.
students.get('students/john')

# Retrieve a document by body with "_id" field.
```

(continues on next page)

(continued from previous page)

```
students.get({'_id': 'students/john'})

# Retrieve a document by body with "_key" field.
students.get({'_key': 'john'})

# Retrieve multiple documents by ID, key or body.
students.get_many(['abby', 'students/lola', {'_key': 'john'}])

# Update a single document.
lola['GPA'] = 2.6
students.update(lola)

# Update one or more matching documents.
students.update_match({'last': 'Park'}, {'GPA': 3.0})

# Replace a single document.
emma['GPA'] = 3.1
students.replace(emma)

# Replace one or more matching documents.
becky = {'first': 'Becky', 'last': 'Solis', 'GPA': '3.3'}
students.replace_match({'first': 'Emma'}, becky)

# Delete a document by key.
students.delete('john')

# Delete a document by ID.
students.delete('students/lola')

# Delete a document by body with "_id" or "_key" field.
students.delete(emma)

# Delete multiple documents. Missing ones are ignored.
students.delete_many([abby, 'john', 'students/lola'])

# Iterate through all documents and update individually.
for student in students:
    student['GPA'] = 4.0
    student['happy'] = True
    students.update(student)
```

You can manage documents via database API wrappers also, but only simple operations (i.e. get, insert, update, replace, delete) are supported and you must provide document IDs instead of keys:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Create some test documents to play around with.
# The documents must have the "_id" field instead.
lola = {'_id': 'students/lola', 'GPA': 3.5}
abby = {'_id': 'students/abby', 'GPA': 3.2}
```

(continues on next page)

(continued from previous page)

```
john = {'_id': 'students/john', 'GPA': 3.6}
emma = {'_id': 'students/emma', 'GPA': 4.0}

# Insert a new document.
metadata = db.insert_document('students', lola)
assert metadata['_id'] == 'students/lola'
assert metadata['_key'] == 'lola'

# Check if a document exists.
assert db.has_document(lola) is True

# Get a document (by ID or body with "_id" field).
db.document('students/lola')
db.document(abby)

# Update a document.
lola['GPA'] = 3.6
db.update_document(lola)

# Replace a document.
lola['GPA'] = 3.4
db.replace_document(lola)

# Delete a document (by ID or body with "_id" field).
db.delete_document('students/lola')
```

See *StandardDatabase* and *StandardCollection* for API specification.

When managing documents, using collection API wrappers over database API wrappers is recommended as more operations are available and less sanity checking is performed under the hood.

3.5 Indexes

Indexes can be added to collections to speed up document lookups. Every collection has a primary hash index on `_key` field by default. This index cannot be deleted or modified. Every edge collection has additional indexes on fields `_from` and `_to`. For more information on indexes, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Create a new collection named "cities".
cities = db.create_collection('cities')

# List the indexes in the collection.
cities.indexes()

# Add a new hash index on document fields "continent" and "country".
index = cities.add_hash_index(fields=['continent', 'country'], unique=True)
```

(continues on next page)

(continued from previous page)

```
# Add new fulltext indexes on fields "continent" and "country".
index = cities.add_fulltext_index(fields=['continent'])
index = cities.add_fulltext_index(fields=['country'])

# Add a new skiplist index on field 'population'.
index = cities.add_skiplist_index(fields=['population'], sparse=False)

# Add a new geo-spatial index on field 'coordinates'.
index = cities.add_geo_index(fields=['coordinates'])

# Add a new persistent index on fields 'currency'.
index = cities.add_persistent_index(fields=['currency'], sparse=True)

# Delete the last index from the collection.
cities.delete_index(index['id'])
```

See *StandardCollection* for API specification.

3.6 Graphs

A **graph** consists of vertices and edges. Vertices are stored as documents in *vertex collections* and edges stored as documents in *edge collections*. The collections used in a graph and their relations are specified with *edge definitions*. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List existing graphs in the database.
db.graphs()

# Create a new graph named "school" if it does not already exist.
# This returns an API wrapper for "school" graph.
if db.has_graph('school'):
    school = db.graph('school')
else:
    school = db.create_graph('school')

# Retrieve various graph properties.
school.name
school.db_name
school.vertex_collections()
school.edge_definitions()

# Delete the graph.
db.delete_graph('school')
```

3.6.1 Edge Definitions

An **edge definition** specifies a directed relation in a graph. A graph can have arbitrary number of edge definitions. Each edge definition consists of the following components:

- **From Vertex Collections:** contain "from" vertices referencing "to" vertices.
- **To Vertex Collections:** contain "to" vertices referenced by "from" vertices.
- **Edge Collection:** contains edges that link "from" and "to" vertices.

Here is an example body of an edge definition:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

Here is an example showing how edge definitions are managed:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
if db.has_graph('school'):
    school = db.graph('school')
else:
    school = db.create_graph('school')

# Create an edge definition named "teach". This creates any missing
# collections and returns an API wrapper for "teach" edge collection.
if not school.has_edge_definition('teach'):
    teach = school.create_edge_definition(
        edge_collection='teach',
        from_vertex_collections=['teachers'],
        to_vertex_collections=['teachers']
    )

# List edge definitions.
school.edge_definitions()

# Replace the edge definition.
school.replace_edge_definition(
    edge_collection='teach',
    from_vertex_collections=['teachers'],
    to_vertex_collections=['lectures']
)

# Delete the edge definition (and its collections).
school.delete_edge_definition('teach', purge=True)
```

3.6.2 Vertex Collections

A **vertex collection** contains vertex documents, and shares its namespace with all other types of collections. Each graph can have an arbitrary number of vertex collections. Vertex collections that are not part of any edge definition are called **orphan collections**. You can manage vertex documents via standard collection API wrappers, but using vertex collection API wrappers provides additional safeguards:

- All modifications are executed in transactions.
- If a vertex is deleted, all connected edges are also automatically deleted.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Create a new vertex collection named "teachers" if it does not exist.
# This returns an API wrapper for "teachers" vertex collection.
if school.has_vertex_collection('teachers'):
    teachers = school.vertex_collection('teachers')
else:
    teachers = school.create_vertex_collection('teachers')

# List vertex collections in the graph.
school.vertex_collections()

# Vertex collections have similar interface as standard collections.
teachers.properties()
teachers.insert({'_key': 'jon', 'name': 'Jon'})
teachers.update({'_key': 'jon', 'age': 35})
teachers.replace({'_key': 'jon', 'name': 'Jon', 'age': 36})
teachers.get('jon')
teachers.has('jon')
teachers.delete('jon')
```

You can manage vertices via graph API wrappers also, but you must use document IDs instead of keys where applicable.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Create a new vertex collection named "lectures" if it does not exist.
# This returns an API wrapper for "lectures" vertex collection.
```

(continues on next page)

(continued from previous page)

```
if school.has_vertex_collection('lectures'):
    school.vertex_collection('lectures')
else:
    school.create_vertex_collection('lectures')

# The "_id" field is required instead of "_key" field (except for insert).
school.insert_vertex('lectures', {'_key': 'CSC101'})
school.update_vertex({'_id': 'lectures/CSC101', 'difficulty': 'easy'})
school.replace_vertex({'_id': 'lectures/CSC101', 'difficulty': 'hard'})
school.has_vertex('lectures/CSC101')
school.vertex('lectures/CSC101')
school.delete_vertex('lectures/CSC101')
```

See *Graph* and *VertexCollection* for API specification.

3.6.3 Edge Collections

An **edge collection** contains *edge documents*, and shares its namespace with all other types of collections. You can manage edge documents via standard collection API wrappers, but using edge collection API wrappers provides additional safeguards:

- All modifications are executed in transactions.
- Edge documents are checked against the edge definitions on insert.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Get the API wrapper for edge collection "teach".
if school.has_edge_definition('teach'):
    teach = school.edge_collection('teach')
else:
    teach = school.create_edge_definition(
        edge_collection='teach',
        from_vertex_collections=['teachers'],
        to_vertex_collections=['lectures']
    )

# Edge collections have a similar interface as standard collections.
teach.insert({
    '_key': 'jon-CSC101',
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101'
})
teach.replace({
    '_key': 'jon-CSC101',
```

(continues on next page)

(continued from previous page)

```
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101',
    'online': False
})
teach.update({
    '_key': 'jon-CSC101',
    'online': True
})
teach.has('jon-CSC101')
teach.get('jon-CSC101')
teach.delete('jon-CSC101')

# Create an edge between two vertices (essentially the same as insert).
teach.link('teachers/jon', 'lectures/CSC101', data={'online': False})

# List edges going in/out of a vertex.
teach.edges('teachers/jon', direction='in')
teach.edges('teachers/jon', direction='out')
```

You can manage edges via graph API wrappers also, but you must use document IDs instead of keys where applicable.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# The "_id" field is required instead of "_key" field.
school.insert_edge(
    collection='teach',
    edge={
        '_id': 'teach/jon-CSC101',
        '_from': 'teachers/jon',
        '_to': 'lectures/CSC101'
    }
)
school.replace_edge({
    '_id': 'teach/jon-CSC101',
    '_from': 'teachers/jon',
    '_to': 'lectures/CSC101',
    'online': False,
})
school.update_edge({
    '_id': 'teach/jon-CSC101',
    'online': True
})
school.has_edge('teach/jon-CSC101')
school.edge('teach/jon-CSC101')
school.delete_edge('teach/jon-CSC101')
school.link('teach', 'teachers/jon', 'lectures/CSC101')
```

(continues on next page)

(continued from previous page)

```
school.edges('teach', 'teachers/jon', direction='in')
```

See *Graph* and *EdgeCollection* for API specification.

3.6.4 Graph Traversals

Graph traversals are executed via the *arango.graph.Graph.traverse()* method. Each traversal can span across multiple vertex collections, and walk over edges and vertices using various algorithms.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for graph "school".
school = db.graph('school')

# Get API wrappers for "from" and "to" vertex collections.
teachers = school.vertex_collection('teachers')
lectures = school.vertex_collection('lectures')

# Get the API wrapper for the edge collection.:
teach = school.edge_collection('teach')

# Insert vertices into the graph.
teachers.insert({'_key': 'jon', 'name': 'Professor jon'})
lectures.insert({'_key': 'CSC101', 'name': 'Introduction to CS'})
lectures.insert({'_key': 'MAT223', 'name': 'Linear Algebra'})
lectures.insert({'_key': 'STA201', 'name': 'Statistics'})

# Insert edges into the graph.
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/CSC101'})
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/STA201'})
teach.insert({'_from': 'teachers/jon', '_to': 'lectures/MAT223'})

# Traverse the graph in outbound direction, breath-first.
school.traverse(
    start_vertex='teachers/jon',
    direction='outbound',
    strategy='bfs',
    edge_uniqueness='global',
    vertex_uniqueness='global',
)
```

See *arango.graph.Graph.traverse()* for API specification.

3.7 AQL

ArangoDB Query Language (AQL) is used to read and write data. It is similar to SQL for relational databases, but without the support for data definition operations such as creating or deleting *databases*, *collections* or *indexes*. For more information, refer to ArangoDB manual.

3.7.1 AQL Queries

AQL queries are invoked from AQL API wrapper. Executing queries returns *result cursors*.

Example:

```
from arango import ArangoClient, AQLQueryKillError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Insert some test documents into "students" collection.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21}
])

# Get the AQL API wrapper.
aql = db.aql

# Retrieve the execution plan without running the query.
aql.explain('FOR doc IN students RETURN doc')

# Validate the query without executing it.
aql.validate('FOR doc IN students RETURN doc')

# Execute the query
cursor = db.aql.execute(
  'FOR doc IN students FILTER doc.age < @value RETURN doc',
  bind_vars={'value': 19}
)
# Iterate through the result cursor
student_keys = [doc['_key'] for doc in cursor]

# List currently running queries.
aql.queries()

# List any slow queries.
aql.slow_queries()

# Clear slow AQL queries if any.
aql.clear_slow_queries()

# Retrieve AQL query tracking properties.
aql.tracking()
```

(continues on next page)

(continued from previous page)

```
# Configure AQL query tracking properties.
aql.set_tracking(
    max_slow_queries=10,
    track_bind_vars=True,
    track_slow_queries=True
)

# Kill a running query (this should fail due to invalid ID).
try:
    aql.kill('some_query_id')
except AQLQueryKillError as err:
    assert err.http_code == 400
    assert err.error_code == 1591
    assert 'cannot kill query' in err.message
```

See *AQL* for API specification.

3.7.2 AQL User Functions

AQL User Functions are custom functions you define in Javascript to extend AQL functionality. They are somewhat similar to SQL procedures.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the AQL API wrapper.
aql = db.aql

# Create a new AQL user function.
aql.create_function(
    # Grouping by name prefix is supported.
    name='functions::temperature::converter',
    code='function (celsius) { return celsius * 1.8 + 32; }'
)
# List AQL user functions.
aql.functions()

# Delete an existing AQL user function.
aql.delete_function('functions::temperature::converter')
```

See *AQL* for API specification.

3.7.3 AQL Query Cache

AQL Query Cache is used to minimize redundant calculation of the same query results. It is useful when read queries are issued frequently and write queries are not.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the AQL API wrapper.
aql = db.aql

# Retrieve AQL query cache properties.
aql.cache.properties()

# Configure AQL query cache properties
aql.cache.configure(mode='demand', limit=10000)

# Clear results in AQL query cache.
aql.cache.clear()
```

See *AQLQueryCache* for API specification.

3.8 Cursors

Many operations provided by python-arango (e.g. executing *AQL* queries) return result **cursors** to batch the network communication between ArangoDB server and python-arango client. Each HTTP request from a cursor fetches the next batch of results (usually documents). Depending on the query, the total number of items in the result set may or may not be known in advance.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Set up some test data to query against.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21},
    {'_key': 'Suzy', 'age': 23},
    {'_key': 'Dave', 'age': 20}
])

# Execute an AQL query which returns a cursor object.
cursor = db.aql.execute(
    'FOR doc IN students FILTER doc.age > @val RETURN doc',
    bind_vars={'val': 17},
    batch_size=2,
    count=True
)
```

(continues on next page)

(continued from previous page)

```
# Get the cursor ID.
cursor.id

# Get the items in the current batch.
cursor.batch()

# Check if the current batch is empty.
cursor.empty()

# Get the total count of the result set.
cursor.count()

# Flag indicating if there are more to be fetched from server.
cursor.has_more()

# Flag indicating if the results are cached.
cursor.cached()

# Get the cursor statistics.
cursor.statistics()

# Get the performance profile.
cursor.profile()

# Get any warnings produced from the query.
cursor.warnings()

# Return the next item from the cursor. If current batch is depleted, the
# next batch if fetched from the server automatically.
cursor.next()

# Return the next item from the cursor. If current batch is depleted, an
# exception is thrown. You need to fetch the next batch manually.
cursor.pop()

# Fetch the next batch and add them to the cursor object.
cursor.fetch()

# Delete the cursor from the server.
cursor.close()
```

See *Cursor* for API specification.

If the fetched result batch is depleted while you are iterating over a cursor (or while calling the method *arango.cursor.Cursor.next()*), python-arango automatically sends an HTTP request to the server to fetch the next batch (just-in-time style). To control exactly when the fetches occur, you can use methods *arango.cursor.Cursor.fetch()* and *arango.cursor.Cursor.pop()* instead.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
```

(continues on next page)

(continued from previous page)

```
db = client.db('test', username='root', password='passwd')

# Set up some test data to query against.
db.collection('students').insert_many([
    {'_key': 'Abby', 'age': 22},
    {'_key': 'John', 'age': 18},
    {'_key': 'Mary', 'age': 21}
])

# If you iterate over the cursor or call cursor.next(), batches are
# fetched automatically from the server just-in-time style.
cursor = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
result = [doc for doc in cursor]

# Alternatively, you can manually fetch and pop for finer control.
cursor = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
while cursor.has_more(): # Fetch until nothing is left on the server.
    cursor.fetch()
while not cursor.empty(): # Pop until nothing is left on the cursor.
    cursor.pop()
```

When running queries in *transactions*, cursors are loaded with the entire result set right away. This is regardless of the parameters passed in when executing the query (e.g. batch_size). You must be mindful of client-side memory capacity when executing queries that can potentially return a large result set.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the total document count in "students" collection.
document_count = db.collection('students').count()

# Execute an AQL query normally (without using transactions).
cursor1 = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)

# Execute the same AQL query in a transaction.
txn_db = db.begin_transaction()
job = txn_db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
txn_db.commit()
cursor2 = job.result()

# The first cursor acts as expected. Its current batch contains only 1 item
# and it still needs to fetch the rest of its result set from the server.
assert len(cursor1.batch()) == 1
assert cursor1.has_more() is True

# The second cursor is pre-loaded with the entire result set, and does not
# require further communication with ArangoDB server. Note that value of
# parameter "batch_size" was ignored.
assert len(cursor2.batch()) == document_count
assert cursor2.has_more() is False
```

3.9 Async Execution

Python-arango supports **async execution**, where it sends requests to ArangoDB server in fire-and-forget style (HTTP 202 returned). The server places incoming requests in its queue and processes them in the background. The results can be retrieved from the server later via *AsyncJob* objects.

Example:

```
import time

from arango import (
    ArangoClient,
    AQLQueryExecuteError,
    AsyncJobCancelError,
    AsyncJobClearError
)

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Begin async execution. This returns an instance of AsyncDatabase, a
# database-level API wrapper tailored specifically for async execution.
async_db = db.begin_async_execution(return_result=True)

# Child wrappers are also tailored for async execution.
async_aql = async_db.aql
async_col = async_db.collection('students')

# API execution context is always set to "async".
assert async_db.context == 'async'
assert async_aql.context == 'async'
assert async_col.context == 'async'

# On API execution, AsyncJob objects are returned instead of results.
job1 = async_col.insert({'_key': 'Neal'})
job2 = async_col.insert({'_key': 'Lily'})
job3 = async_aql.execute('RETURN 100000')
job4 = async_aql.execute('INVALID QUERY')  # Fails due to syntax error.

# Retrieve the status of each async job.
for job in [job1, job2, job3, job4]:
    # Job status can be "pending", "done" or "cancelled".
    assert job.status() in {'pending', 'done', 'cancelled'}

    # Let's wait until the jobs are finished.
    while job.status() != 'done':
        time.sleep(0.1)

# Retrieve the results of successful jobs.
metadata = job1.result()
assert metadata['_id'] == 'students/Neal'

metadata = job2.result()
assert metadata['_id'] == 'students/Lily'
```

(continues on next page)

(continued from previous page)

```
cursor = job3.result()
assert cursor.next() == 100000

# If a job fails, the exception is propagated up during result retrieval.
try:
    result = job4.result()
except AQLQueryExecuteError as err:
    assert err.http_code == 400
    assert err.error_code == 1501
    assert 'syntax error' in err.message

# Cancel a job. Only pending jobs still in queue may be cancelled.
# Since job3 is done, there is nothing to cancel and an exception is raised.
try:
    job3.cancel()
except AsyncJobCancelError as err:
    assert err.message.endswith('job {} not found'.format(job3.id))

# Clear the result of a job from ArangoDB server to free up resources.
# Result of job4 was removed from the server automatically upon retrieval,
# so attempt to clear it raises an exception.
try:
    job4.clear()
except AsyncJobClearError as err:
    assert err.message.endswith('job {} not found'.format(job4.id))

# List the IDs of the first 100 async jobs completed.
db.async_jobs(status='done', count=100)

# List the IDs of the first 100 async jobs still pending.
db.async_jobs(status='pending', count=100)

# Clear all async jobs still sitting on the server.
db.clear_async_jobs()
```

Note: Be mindful of server-side memory capacity when issuing a large number of async requests in small time interval.

See *AsyncDatabase* and *AsyncJob* for API specification.

3.10 Batch Execution

Python-arango supports **batch execution**. Requests to ArangoDB server are placed in client-side in-memory queue, and committed together in a single HTTP call. After the commit, results can be retrieved from *BatchJob* objects.

Example:

```
from arango import ArangoClient, AQLQueryExecuteError

# Initialize the ArangoDB client.
client = ArangoClient()
```

(continues on next page)

(continued from previous page)

```
# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Begin batch execution via context manager. This returns an instance of
# BatchDatabase, a database-level API wrapper tailored specifically for
# batch execution. The batch is automatically committed when exiting the
# context. The BatchDatabase wrapper cannot be reused after commit.
with db.begin_batch_execution(return_result=True) as batch_db:

    # Child wrappers are also tailored for batch execution.
    batch_aql = batch_db.aql
    batch_col = batch_db.collection('students')

    # API execution context is always set to "batch".
    assert batch_db.context == 'batch'
    assert batch_aql.context == 'batch'
    assert batch_col.context == 'batch'

    # BatchJob objects are returned instead of results.
    job1 = batch_col.insert({'_key': 'Kris'})
    job2 = batch_col.insert({'_key': 'Rita'})
    job3 = batch_aql.execute('RETURN 100000')
    job4 = batch_aql.execute('INVALID QUERY')  # Fails due to syntax error.

# Upon exiting context, batch is automatically committed.
assert 'Kris' in students
assert 'Rita' in students

# Retrieve the status of each batch job.
for job in batch_db.queued_jobs():
    # Status is set to either "pending" (transaction is not committed yet
    # and result is not available) or "done" (transaction is committed and
    # result is available).
    assert job.status() in {'pending', 'done'}

# Retrieve the results of successful jobs.
metadata = job1.result()
assert metadata['_id'] == 'students/Kris'

metadata = job2.result()
assert metadata['_id'] == 'students/Rita'

cursor = job3.result()
assert cursor.next() == 100000

# If a job fails, the exception is propagated up during result retrieval.
try:
    result = job4.result()
except AQLQueryExecuteError as err:
    assert err.http_code == 400
    assert err.error_code == 1501
    assert 'syntax error' in err.message

# Batch execution can be initiated without using a context manager.
```

(continues on next page)

(continued from previous page)

```
# If return_result parameter is set to False, no jobs are returned.
batch_db = db.begin_batch_execution(return_result=False)
batch_db.collection('students').insert({'_key': 'Jake'})
batch_db.collection('students').insert({'_key': 'Jill'})

# The commit must be called explicitly.
batch_db.commit()
assert 'Jake' in students
assert 'Jill' in students
```

Note:

- Be mindful of client-side memory capacity when issuing a large number of requests in single batch execution.
- *BatchDatabase* and *BatchJob* instances are stateful objects, and should not be shared across multiple threads.
- *BatchDatabase* instance cannot be reused after commit.

See *BatchDatabase* and *BatchJob* for API specification.

3.11 Transactions

Python-arango supports **transactions**, where requests to ArangoDB server are placed in client-side in-memory queue, and committed as a single, logical unit of work (ACID compliant). After a successful commit, results can be retrieved from *TransactionJob* objects.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Begin a transaction via context manager. This returns an instance of
# TransactionDatabase, a database-level API wrapper tailored specifically
# for executing transactions. The transaction is automatically committed
# when exiting the context. The TransactionDatabase wrapper cannot be
# reused after commit and may be discarded after.
with db.begin_transaction() as txn_db:

    # Child wrappers are also tailored for transactions.
    txn_col = txn_db.collection('students')

    # API execution context is always set to "transaction".
    assert txn_db.context == 'transaction'
    assert txn_col.context == 'transaction'

    # TransactionJob objects are returned instead of results.
```

(continues on next page)

(continued from previous page)

```
    job1 = txn_col.insert({'_key': 'Abby'})
    job2 = txn_col.insert({'_key': 'John'})
    job3 = txn_col.insert({'_key': 'Mary'})

# Upon exiting context, transaction is automatically committed.
assert 'Abby' in students
assert 'John' in students
assert 'Mary' in students

# Retrieve the status of each transaction job.
for job in txn_db.queued_jobs():
    # Status is set to either "pending" (transaction is not committed yet
    # and result is not available) or "done" (transaction is committed and
    # result is available).
    assert job.status() in {'pending', 'done'}

# Retrieve the job results.
metadata = job1.result()
assert metadata['_id'] == 'students/Abby'

metadata = job2.result()
assert metadata['_id'] == 'students/John'

metadata = job3.result()
assert metadata['_id'] == 'students/Mary'

# Transactions can be initiated without using a context manager.
# If return_result parameter is set to False, no jobs are returned.
txn_db = db.begin_transaction(return_result=False)
txn_db.collection('students').insert({'_key': 'Jake'})
txn_db.collection('students').insert({'_key': 'Jill'})

# The commit must be called explicitly.
txn_db.commit()
assert 'Jake' in students
assert 'Jill' in students
```

Note:

- Be mindful of client-side memory capacity when issuing a large number of requests in a single transaction.
- *TransactionDatabase* and *TransactionJob* instances are stateful objects, and should not be shared across multiple threads.
- *TransactionDatabase* instance cannot be reused after commit.

See *TransactionDatabase* and *TransactionJob* for API specification.

3.11.1 Error Handling

Unlike *batch* or *async* execution, job-specific error handling is not possible for transactions. As soon as a job fails, the entire transaction is halted, all previous successful jobs are rolled back, and *arango.exceptions.TransactionExecuteError* is raised. The exception describes the first failed job, and all *TransactionJob* objects are left at "pending" status (they may be discarded).

Example:

```
from arango import ArangoClient, TransactionExecuteError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')

# Begin a new transaction.
txn_db = db.begin_transaction()
txn_col = txn_db.collection('students')

job1 = txn_col.insert({'_key': 'Karl'})  # Is going to be rolled back.
job2 = txn_col.insert({'_key': 'Karl'})  # Fails due to duplicate key.
job3 = txn_col.insert({'_key': 'Josh'})  # Never executed on the server.

try:
    txn_db.commit()
except TransactionExecuteError as err:
    assert err.http_code == 409
    assert err.error_code == 1210
    assert err.message.endswith('conflicting key: Karl')

# All operations in the transaction are rolled back.
assert 'Karl' not in students
assert 'Josh' not in students

# All transaction jobs are left at "pending "status and may be discarded.
for job in txn_db.queued_jobs():
    assert job.status() == 'pending'
```

3.11.2 Restrictions

This section covers important restrictions that you must keep in mind before choosing to use transactions.

TransactionJob results are available only *after* commit, and are not accessible during execution. If you need to implement a logic which depends on intermediate, in-transaction values, you can instead call the method `arango.database.Database.execute_transaction()` which takes raw Javascript command as its argument.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')
```

(continues on next page)

(continued from previous page)

```
# Execute transaction in raw Javascript.
result = db.execute_transaction(
    command='''
    function () {{
        var db = require('internal').db;
        db.students.save(params.student1);
        if (db.students.count() > 1) {
            db.students.save(params.student2);
        } else {
            db.students.save(params.student3);
        }
        return true;
    }}
    ''',
    params={
        'student1': {'_key': 'Lucy'},
        'student2': {'_key': 'Greg'},
        'student3': {'_key': 'Dona'}
    },
    read='students',  # Specify the collections read.
    write='students'  # Specify the collections written.
)
assert result is True
assert 'Lucy' in students
assert 'Greg' in students
assert 'Dona' not in students
```

Note that in above example, `arango.database.Database.execute_transaction()` requires names of *read* and *write* collections as python-arango has no way of reliably figuring out which collections are used. This is also the case when executing AQL queries.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Begin a new transaction via context manager.
with db.begin_transaction() as txn_db:
    job = txn_db.aql.execute(
        'INSERT {_key: "Judy", age: @age} IN students RETURN true',
        bind_vars={'age': 19},
        # You must specify the "read" and "write" collections.
        read_collections=[],
        write_collections=['students']
    )
cursor = job.result()
assert cursor.next() is True
assert db.collection('students').get('Judy')['age'] == 19
```

Due to limitations of ArangoDB's REST API, only the following methods are supported in transactions:

- *arango.aql.AQL.execute()*

- *arango.collection.StandardCollection.get()*
- *arango.collection.StandardCollection.get_many()*
- *arango.collection.StandardCollection.insert()*
- *arango.collection.StandardCollection.insert_many()*
- *arango.collection.StandardCollection.update()*
- *arango.collection.StandardCollection.update_many()*
- *arango.collection.StandardCollection.update_match()*
- *arango.collection.StandardCollection.replace()*
- *arango.collection.StandardCollection.replace_many()*
- *arango.collection.StandardCollection.replace_match()*
- *arango.collection.StandardCollection.delete()*
- *arango.collection.StandardCollection.delete_many()*
- *arango.collection.StandardCollection.delete_match()*
- *arango.collection.StandardCollection.properties()*
- *arango.collection.StandardCollection.statistics()*
- *arango.collection.StandardCollection.revision()*
- *arango.collection.StandardCollection.checksum()*
- *arango.collection.StandardCollection.rotate()*
- *arango.collection.StandardCollection.truncate()*
- *arango.collection.StandardCollection.count()*
- *arango.collection.StandardCollection.has()*
- *arango.collection.StandardCollection.ids()*
- *arango.collection.StandardCollection.keys()*
- *arango.collection.StandardCollection.all()*
- *arango.collection.StandardCollection.find()*
- *arango.collection.StandardCollection.find_near()*
- *arango.collection.StandardCollection.find_in_range()*
- *arango.collection.StandardCollection.find_in_radius()*
- *arango.collection.StandardCollection.find_in_box()*
- *arango.collection.StandardCollection.find_by_text()*
- *arango.collection.StandardCollection.get_many()*
- *arango.collection.StandardCollection.random()*
- *arango.collection.StandardCollection.indexes()*
- *arango.collection.VertexCollection.get()*
- *arango.collection.VertexCollection.insert()*
- *arango.collection.VertexCollection.update()*

- *arango.collection.VertexCollection.replace()*
- *arango.collection.VertexCollection.delete()*
- *arango.collection.EdgeCollection.get()*
- *arango.collection.EdgeCollection.insert()*
- *arango.collection.EdgeCollection.update()*
- *arango.collection.EdgeCollection.replace()*
- *arango.collection.EdgeCollection.delete()*

If an unsupported method is called, *arango.exceptions.TransactionStateError* is raised.

Example:

```
from arango import ArangoClient, TransactionStateError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Begin a new transaction.
txn_db = db.begin_transaction()

# API method "databases()" is not supported and an exception is raised.
try:
    txn_db.databases()
except TransactionStateError as err:
    assert err.source == 'client'
    assert err.message == 'action not allowed in transaction'
```

When running queries in transactions, the *cursors* are loaded with the entire result set right away. This is regardless of the parameters passed in when executing the query (e.g batch_size). You must be mindful of client-side memory capacity when executing queries that can potentially return a large result set.

Example:

```
# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the total document count in "students" collection.
document_count = db.collection('students').count()

# Execute an AQL query normally (without using transactions).
cursor1 = db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)

# Execute the same AQL query in a transaction.
with db.begin_transaction() as txn_db:
    job = txn_db.aql.execute('FOR doc IN students RETURN doc', batch_size=1)
cursor2 = job.result()

# The first cursor acts as expected. Its current batch contains only 1 item
# and it still needs to fetch the rest of its result set from the server.
```

(continues on next page)

(continued from previous page)

```
assert len(cursor1.batch()) == 1
assert cursor1.has_more() is True

# The second cursor is pre-loaded with the entire result set, and does not
# require further communication with ArangoDB server. Note that value of
# parameter "batch_size" was ignored.
assert len(cursor2.batch()) == document_count
assert cursor2.has_more() is False
```

3.12 Server Administration

Python-arango provides operations for server administration and monitoring. Most of these operations can only be performed by admin users via `_system` database.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db('_system', username='root', password='passwd')

# Check the server connection by sending a test GET request.
sys_db.ping()

# Retrieve the server version.
sys_db.version()

# Retrieve the server details.
sys_db.details()

# Retrieve the target DB version.
sys_db.required_db_version()

# Retrieve the database engine.
sys_db.engine()

# Retrieve the server time.
sys_db.time()

# Retrieve the server role in a cluster.
sys_db.role()

# Retrieve the server statistics.
sys_db.statistics()

# Read the server log.
sys_db.read_log(level="debug")

# Retrieve the log levels.
sys_db.log_levels()
```

(continues on next page)

(continued from previous page)

```
# Set the log .
sys_db.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)

# Echo the last request.
sys_db.echo()

# Reload the routing collection.
sys_db.reload_routing()
```

See *StandardDatabase* for API specification.

3.13 Users and Permissions

Python-arango provides operations for managing users and permissions. Most of these operations can only be performed by admin users via `_system` database.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
sys_db = client.db('_system', username='root', password='passwd')

# List all users.
sys_db.users()

# Create a new user.
sys_db.create_user(
    username='johndoe@gmail.com',
    password='first_password',
    active=True,
    extra={'team': 'backend', 'title': 'engineer'}
)

# Check if a user exists.
sys_db.has_user('johndoe@gmail.com')

# Retrieve details of a user.
sys_db.user('johndoe@gmail.com')

# Update an existing user.
sys_db.update_user(
    username='johndoe@gmail.com',
    password='second_password',
    active=True,
    extra={'team': 'frontend', 'title': 'engineer'}
)
```

(continues on next page)

(continued from previous page)

```
# Replace an existing user.
sys_db.replace_user(
    username='johndoe@gmail.com',
    password='third_password',
    active=True,
    extra={'team': 'frontend', 'title': 'architect'}
)

# Retrieve user permissions for all databases and collections.
sys_db.permissions('johndoe@gmail.com')

# Retrieve user permission for "test" database.
sys_db.permission(
    username='johndoe@gmail.com',
    database='test'
)

# Retrieve user permission for "students" collection in "test" database.
sys_db.permission(
    username='johndoe@gmail.com',
    database='test',
    collection='students'
)

# Update user permission for "test" database.
sys_db.update_permission(
    username='johndoe@gmail.com',
    permission='rw',
    database='test'
)

# Update user permission for "students" collection in "test" database.
sys_db.update_permission(
    username='johndoe@gmail.com',
    permission='ro',
    database='test',
    collection='students'
)

# Reset user permission for "test" database.
sys_db.reset_permission(
    username='johndoe@gmail.com',
    database='test'
)

# Reset user permission for "students" collection in "test" database.
sys_db.reset_permission(
    username='johndoe@gmail.com',
    database='test',
    collection='students'
)
```

See *StandardDatabase* for API specification.

3.14 Tasks

ArangoDB can schedule user-defined Javascript snippets as one-time or periodic (re-scheduled after each execution) tasks. Tasks are executed in the context of the database they are defined in.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# List all active tasks
db.tasks()

# Create a new task which simply prints parameters.
db.create_task(
    name='test_task',
    command='''
        var task = function(params){
            var db = require('@arangodb');
            db.print(params);
        }
        task(params);
    ''',
    params={'foo': 'bar'},
    offset=300,
    period=10,
    task_id='001'
)

# Retrieve details on a task by ID.
db.task('001')

# Delete an existing task by ID.
db.delete_task('001', ignore_missing=True)
```

Note: When deleting a database, any tasks that were initialized under its context remain active. It is therefore advisable to delete any running tasks before deleting the database.

Refer to *StandardDatabase* class for API specification.

3.15 Write-Ahead Log

Write-Ahead Log (WAL) is a set of append-only files recording all writes on ArangoDB server. It is typically used to perform data recovery after a crash or synchronize slave databases with master databases in replicated environments. WAL operations can only be performed by admin users via `_system` database.

Example:

See `WriteAheadLog` for API specification.

3.16 Pregel

Python-arango supports **Pregel**, an ArangoDB module for distributed iterative graph processing. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the Pregel API wrapper.
pregel = db.pregel

# Start a new Pregel job in "school" graph.
job_id = db.pregel.create_job(
    graph='school',
    algorithm='pagerank',
    store=False,
    max_gss=100,
    thread_count=1,
    async_mode=False,
    result_field='result',
    algorithm_params={'threshold': 0.000001}
)

# Retrieve details of a Pregel job by ID.
job = pregel.job(job_id)

# Delete a Pregel job by ID.
pregel.delete_job(job_id)
```

See *Pregel* for API specification.

3.17 Foxx

Python-arango supports **Foxx**, a microservice framework which lets you define custom HTTP endpoints to extend ArangoDB's REST API. For more information, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "_system" database as root user.
db = client.db('_system', username='root', password='passwd')

# Get the Foxx API wrapper.
foxx = db.foxx
```

(continues on next page)

(continued from previous page)

```
# Define the test mount point.
service_mount = '/test_mount'

# List services.
foxx.services()

# Create a service.
foxx.create_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    development=True,
    setup=True,
    legacy=True
)

# Update (upgrade) a service.
service = db.foxx.update_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    teardown=True,
    setup=True,
    legacy=False
)

# Replace (overwrite) a service.
service = db.foxx.replace_service(
    mount=service_mount,
    source='/tmp/service.zip',
    config={},
    dependencies={},
    teardown=True,
    setup=True,
    legacy=True,
    force=False
)

# Get service details.
foxx.service(service_mount)

# Manage service configuration.
foxx.config(service_mount)
foxx.update_config(service_mount, config={})
foxx.replace_config(service_mount, config={})

# Manage service dependencies.
foxx.dependencies(service_mount)
foxx.update_dependencies(service_mount, dependencies={})
foxx.replace_dependencies(service_mount, dependencies={})

# Toggle development mode for a service.
foxx.enable_development(service_mount)
foxx.disable_development(service_mount)
```

(continues on next page)

(continued from previous page)

```
# Other miscellaneous functions.
foxx.readme(service_mount)
foxx.swagger(service_mount)
foxx.download(service_mount)
foxx.commit(service_mount)
foxx.scripts(service_mount)
foxx.run_script(service_mount, 'setup', [])
foxx.run_tests(service_mount, reporter='xunit', output_format='xml')

# Delete a service.
foxx.delete_service(service_mount)
```

See *Foxx* for API specification.

3.18 Views

Python-arango supports **view** management. For more information on view properties, refer to ArangoDB manual.

Example:

```
from arango import ArangoClient

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Retrieve list of views.
db.views()

# Create a view.
db.create_view(
    name='foo',
    view_type='arangosearch',
    properties={
        'cleanupIntervalStep': 0,
        'consolidationIntervalMsec': 0
    }
)

# Rename a view.
db.rename_view('foo', 'bar')

# Retrieve view properties.
db.view('bar')

# Partially update view properties.
db.update_view(
    name='bar',
    properties={
        'cleanupIntervalStep': 1000,
        'consolidationIntervalMsec': 200
    }
```

(continues on next page)

(continued from previous page)

```
)

# Replace view properties. Unspecified ones are reset to default.
db.replace_view(
    name='bar',
    properties={'cleanupIntervalStep': 2000}
)

# Delete a view.
db.delete_view('bar')
```

Refer to *StandardDatabase* class for API specification.

3.19 Threading

Instances of the following classes are considered *stateful*, and should not be shared across multiple threads without locks in place:

- *BatchDatabase* (see *Batch Execution*)
- *BatchJob* (see *Batch Execution*)
- *Cursor* (see *Cursors*)
- *TransactionDatabase* (see *Transactions*)
- *TransactionJob* (see *Transactions*)

The rest of python-arango is safe to use in multi-threaded environments.

3.20 Error Handling

All python-arango exceptions inherit *arango.exceptions.ArangoError*, which splits into subclasses *arango.exceptions.ArangoServerError* and *arango.exceptions.ArangoClientError*.

3.20.1 Server Errors

arango.exceptions.ArangoServerError exceptions lightly wrap non-2xx HTTP responses coming from ArangoDB. Each exception object contains the error message, error code and HTTP request response details.

Example:

```
from arango import ArangoClient, ArangoServerError, DocumentInsertError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')

# Get the API wrapper for "students" collection.
students = db.collection('students')
```

(continues on next page)

(continued from previous page)

```
try:
    students.insert({'_key': 'John'})
    students.insert({'_key': 'John'})  # duplicate key error

except DocumentInsertError as exc:

    assert isinstance(exc, ArangoServerError)
    assert exc.source == 'server'

    exc.message           # Exception message usually from ArangoDB
    exc.error_message     # Raw error message from ArangoDB
    exc.error_code        # Error code from ArangoDB
    exc.url               # URL (API endpoint)
    exc.http_method       # HTTP method (e.g. "POST")
    exc.http_headers      # Response headers
    exc.http_code         # Status code (e.g. 200)

    # You can inspect the ArangoDB response directly.
    response = exc.response
    response.method       # HTTP method (e.g. "POST")
    response.headers      # Response headers
    response.url          # Full request URL
    response.is_success   # Set to True if HTTP code is 2XX
    response.body         # JSON-deserialized response body
    response.raw_body     # Raw string response body
    response.status_text  # Status text (e.g "OK")
    response.status_code  # Status code (e.g. 200)
    response.error_code   # Error code from ArangoDB

    # You can also inspect the request sent to ArangoDB.
    request = exc.request
    request.method        # HTTP method (e.g. "post")
    request.endpoint      # API endpoint starting with "/_api"
    request.headers       # Request headers
    request.params        # URL parameters
    request.data          # Request payload
    request.read          # Read collections (used for transactions only)
    request.write         # Write collections (used for transactions only)
    request.command       # ArangoSh command (used for transactions only)
```

See *Response* and *Request* for reference.

3.20.2 Client Errors

arango.exceptions.ArangoClientError exceptions originate from python-arango client itself. They do not contain error codes nor HTTP request response details.

Example:

```
from arango import ArangoClient, ArangoClientError, DocumentParseError

# Initialize the ArangoDB client.
client = ArangoClient()

# Connect to "test" database as root user.
db = client.db('test', username='root', password='passwd')
```

(continues on next page)

(continued from previous page)

```
# Get the API wrapper for "students" collection.
students = db.collection('students')

try:
    students.get({'_id': 'invalid_id'})  # malformed document

except DocumentParseError as exc:

    assert isinstance(exc, ArangoClientError)
    assert exc.source == 'client'

    # Only the error message is set.
    error_message = exc.message
    assert exc.error_code is None
    assert exc.error_message is None
    assert exc.url is None
    assert exc.http_method is None
    assert exc.http_code is None
    assert exc.http_headers is None
    assert exc.response is None
    assert exc.request is None
```

3.20.3 Exceptions

Below are all exceptions from python-arango.

exception arango.exceptions.**AQLCacheClearError**(*resp*, *request*, *msg=None*)
Failed to clear the query cache.

exception arango.exceptions.**AQLCacheConfigureError**(*resp*, *request*, *msg=None*)
Failed to configure query cache properties.

exception arango.exceptions.**AQLCacheEntriesError**(*resp*, *request*, *msg=None*)
Failed to retrieve AQL cache entries.

exception arango.exceptions.**AQLCachePropertiesError**(*resp*, *request*, *msg=None*)
Failed to retrieve query cache properties.

exception arango.exceptions.**AQLFunctionCreateError**(*resp*, *request*, *msg=None*)
Failed to create AQL user function.

exception arango.exceptions.**AQLFunctionDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete AQL user function.

exception arango.exceptions.**AQLFunctionListError**(*resp*, *request*, *msg=None*)
Failed to retrieve AQL user functions.

exception arango.exceptions.**AQLQueryClearError**(*resp*, *request*, *msg=None*)
Failed to clear slow AQL queries.

exception arango.exceptions.**AQLQueryExecuteError**(*resp*, *request*, *msg=None*)
Failed to execute query.

exception arango.exceptions.**AQLQueryExplainError**(*resp*, *request*, *msg=None*)
Failed to parse and explain query.

exception arango.exceptions.**AQLQueryKillError**(*resp*, *request*, *msg=None*)
Failed to kill the query.

exception arango.exceptions.**AQLQueryListError**(*resp*, *request*, *msg=None*)
Failed to retrieve running AQL queries.

exception arango.exceptions.**AQLQueryTrackingGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve AQL tracking properties.

exception arango.exceptions.**AQLQueryTrackingSetError**(*resp*, *request*, *msg=None*)
Failed to configure AQL tracking properties.

exception arango.exceptions.**AQLQueryValidateError**(*resp*, *request*, *msg=None*)
Failed to parse and validate query.

exception arango.exceptions.**ArangoClientError**(*msg*)
Base class for errors originating from python-arango client.

Parameters **msg** (*str | unicode*) – Error message.

Variables

- **source** (*str | unicode*) – Source of the error (always set to "client").
- **message** (*str | unicode*) – Error message.

exception arango.exceptions.**ArangoError**
Base class for all exceptions in python-arango.

exception arango.exceptions.**ArangoServerError**(*resp*, *request*, *msg=None*)
Base class for errors originating from ArangoDB server.

Parameters

- **resp** (arango.response.Response) – HTTP response.
- **msg** (*str | unicode*) – Error message override.

Variables

- **source** (*str | unicode*) – Source of the error (always set to "server").
- **message** (*str | unicode*) – Exception message.
- **url** (*str | unicode*) – API URL.
- **response** (arango.response.Response) – HTTP response object.
- **request** (arango.request.Request) – HTTP request object.
- **http_method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **http_code** (*int*) – HTTP status code.
- **http_headers** (*requests.structures.CaseInsensitiveDict | dict*) – Response headers.
- **error_code** (*int*) – Error code from ArangoDB server.
- **error_message** (*str | unicode*) – Raw error message from ArangoDB server.

exception arango.exceptions.**AsyncExecuteError**(*resp*, *request*, *msg=None*)
Failed to execute async API request.

exception arango.exceptions.**AsyncJobCancelError**(*resp*, *request*, *msg=None*)
Failed to cancel async job.

exception arango.exceptions.**AsyncJobClearError**(*resp*, *request*, *msg=None*)
Failed to clear async job results.

exception arango.exceptions.**AsyncJobListError**(*resp*, *request*, *msg=None*)
Failed to retrieve async jobs.

exception arango.exceptions.**AsyncJobResultError**(*resp*, *request*, *msg=None*)
Failed to retrieve async job result.

exception arango.exceptions.**AsyncJobStatusError**(*resp*, *request*, *msg=None*)
Failed to retrieve async job status.

exception arango.exceptions.**BatchExecuteError**(*resp*, *request*, *msg=None*)
Failed to execute batch API request.

exception arango.exceptions.**BatchJobResultError**(*msg*)
Failed to retrieve batch job result.

exception arango.exceptions.**BatchStateError**(*msg*)
The batch object was in a bad state.

exception arango.exceptions.**CollectionChecksumError**(*resp*, *request*, *msg=None*)
Failed to retrieve collection checksum.

exception arango.exceptions.**CollectionConfigureError**(*resp*, *request*, *msg=None*)
Failed to configure collection properties.

exception arango.exceptions.**CollectionCreateError**(*resp*, *request*, *msg=None*)
Failed to create collection.

exception arango.exceptions.**CollectionDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete collection.

exception arango.exceptions.**CollectionListError**(*resp*, *request*, *msg=None*)
Failed to retrieve collections.

exception arango.exceptions.**CollectionLoadError**(*resp*, *request*, *msg=None*)
Failed to load collection.

exception arango.exceptions.**CollectionPropertiesError**(*resp*, *request*, *msg=None*)
Failed to retrieve collection properties.

exception arango.exceptions.**CollectionRenameError**(*resp*, *request*, *msg=None*)
Failed to rename collection.

exception arango.exceptions.**CollectionRevisionError**(*resp*, *request*, *msg=None*)
Failed to retrieve collection revision.

exception arango.exceptions.**CollectionRotateJournalError**(*resp*, *request*, *msg=None*)
Failed to rotate collection journal.

exception arango.exceptions.**CollectionStatisticsError**(*resp*, *request*, *msg=None*)
Failed to retrieve collection statistics.

exception arango.exceptions.**CollectionTruncateError**(*resp*, *request*, *msg=None*)
Failed to truncate collection.

exception arango.exceptions.**CollectionUnloadError**(*resp*, *request*, *msg=None*)
Failed to unload collection.

exception arango.exceptions.**CursorCloseError**(*resp*, *request*, *msg=None*)
Failed to delete the cursor result from server.

exception arango.exceptions.**CursorEmptyError**(*msg*)
The current batch in cursor was empty.

exception arango.exceptions.**CursorNextError**(*resp*, *request*, *msg=None*)
Failed to retrieve the next result batch from server.

exception arango.exceptions.**CursorStateError**(*msg*)
The cursor object was in a bad state.

exception arango.exceptions.**DatabaseCreateError**(*resp*, *request*, *msg=None*)
Failed to create database.

exception arango.exceptions.**DatabaseDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete database.

exception arango.exceptions.**DatabaseListError**(*resp*, *request*, *msg=None*)
Failed to retrieve databases.

exception arango.exceptions.**DatabasePropertiesError**(*resp*, *request*, *msg=None*)
Failed to retrieve database properties.

exception arango.exceptions.**DocumentCountError**(*resp*, *request*, *msg=None*)
Failed to retrieve document count.

exception arango.exceptions.**DocumentDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete document.

exception arango.exceptions.**DocumentGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve document.

exception arango.exceptions.**DocumentIDsError**(*resp*, *request*, *msg=None*)
Failed to retrieve document IDs.

exception arango.exceptions.**DocumentInError**(*resp*, *request*, *msg=None*)
Failed to check whether document exists.

exception arango.exceptions.**DocumentInsertError**(*resp*, *request*, *msg=None*)
Failed to insert document.

exception arango.exceptions.**DocumentKeysError**(*resp*, *request*, *msg=None*)
Failed to retrieve document keys.

exception arango.exceptions.**DocumentParseError**(*msg*)
Failed to parse document input.

exception arango.exceptions.**DocumentReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace document.

exception arango.exceptions.**DocumentRevisionError**(*resp*, *request*, *msg=None*)
The expected and actual document revisions mismatched.

exception arango.exceptions.**DocumentUpdateError**(*resp*, *request*, *msg=None*)
Failed to update document.

exception arango.exceptions.**EdgeDefinitionCreateError**(*resp*, *request*, *msg=None*)
Failed to create edge definition.

exception arango.exceptions.**EdgeDefinitionDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete edge definition.

exception arango.exceptions.**EdgeDefinitionListError**(*resp*, *request*, *msg=None*)
Failed to retrieve edge definitions.

exception arango.exceptions.**EdgeDefinitionReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace edge definition.

exception arango.exceptions.**EdgeListError**(*resp*, *request*, *msg=None*)
Failed to retrieve edges coming in and out of a vertex.

exception arango.exceptions.**FoxxCommitError**(*resp*, *request*, *msg=None*)
Failed to commit local Foxx service state.

exception arango.exceptions.**FoxxConfigGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service configuration.

exception arango.exceptions.**FoxxConfigReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace Foxx service configuration.

exception arango.exceptions.**FoxxConfigUpdateError**(*resp*, *request*, *msg=None*)
Failed to update Foxx service configuration.

exception arango.exceptions.**FoxxDependencyGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service dependencies.

exception arango.exceptions.**FoxxDependencyReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace Foxx service dependencies.

exception arango.exceptions.**FoxxDependencyUpdateError**(*resp*, *request*, *msg=None*)
Failed to update Foxx service dependencies.

exception arango.exceptions.**FoxxDevModeDisableError**(*resp*, *request*, *msg=None*)
Failed to disable development mode for Foxx service.

exception arango.exceptions.**FoxxDevModeEnableError**(*resp*, *request*, *msg=None*)
Failed to enable development mode for Foxx service.

exception arango.exceptions.**FoxxDownloadError**(*resp*, *request*, *msg=None*)
Failed to download Foxx service bundle.

exception arango.exceptions.**FoxxReadmeGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service readme.

exception arango.exceptions.**FoxxScriptListError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service scripts.

exception arango.exceptions.**FoxxScriptRunError**(*resp*, *request*, *msg=None*)
Failed to run Foxx service script.

exception arango.exceptions.**FoxxServiceCreateError**(*resp*, *request*, *msg=None*)
Failed to create Foxx service.

exception arango.exceptions.**FoxxServiceDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete Foxx services.

exception arango.exceptions.**FoxxServiceGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service metadata.

exception arango.exceptions.**FoxxServiceListError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx services.

exception arango.exceptions.**FoxxServiceReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace Foxx service.

exception arango.exceptions.**FoxxServiceUpdateError**(*resp*, *request*, *msg=None*)
Failed to update Foxx service.

exception arango.exceptions.**FoxxSwaggerGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Foxx service swagger.

exception arango.exceptions.**FoxxTestRunError**(*resp*, *request*, *msg=None*)
Failed to run Foxx service tests.

exception arango.exceptions.**GraphCreateError**(*resp*, *request*, *msg=None*)
Failed to create the graph.

exception arango.exceptions.**GraphDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete the graph.

exception arango.exceptions.**GraphListError**(*resp*, *request*, *msg=None*)
Failed to retrieve graphs.

exception arango.exceptions.**GraphPropertiesError**(*resp*, *request*, *msg=None*)
Failed to retrieve graph properties.

exception arango.exceptions.**GraphTraverseError**(*resp*, *request*, *msg=None*)
Failed to execute graph traversal.

exception arango.exceptions.**IndexCreateError**(*resp*, *request*, *msg=None*)
Failed to create collection index.

exception arango.exceptions.**IndexDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete collection index.

exception arango.exceptions.**IndexListError**(*resp*, *request*, *msg=None*)
Failed to retrieve collection indexes.

exception arango.exceptions.**IndexLoadError**(*resp*, *request*, *msg=None*)
Failed to load indexes into memory.

exception arango.exceptions.**PermissionGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve user permission.

exception arango.exceptions.**PermissionListError**(*resp*, *request*, *msg=None*)
Failed to list user permissions.

exception arango.exceptions.**PermissionResetError**(*resp*, *request*, *msg=None*)
Failed to reset user permission.

exception arango.exceptions.**PermissionUpdateError**(*resp*, *request*, *msg=None*)
Failed to update user permission.

exception arango.exceptions.**PregelJobCreateError**(*resp*, *request*, *msg=None*)
Failed to create Pregel job.

exception arango.exceptions.**PregelJobDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete Pregel job.

exception arango.exceptions.**PregelJobGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve Pregel job details.

exception arango.exceptions.**ServerConnectionError**(*msg*)
Failed to connect to ArangoDB server.

exception arango.exceptions.**ServerDetailsError**(*resp*, *request*, *msg=None*)
Failed to retrieve server details.

exception arango.exceptions.**ServerEchoError**(*resp*, *request*, *msg=None*)
Failed to retrieve details on last request.

exception arango.exceptions.**ServerEndpointsError**(*resp*, *request*, *msg=None*)
Failed to retrieve server endpoints.

exception arango.exceptions.**ServerEngineError**(*resp*, *request*, *msg=None*)
Failed to retrieve database engine.

exception arango.exceptions.**ServerLogLevelError**(*resp*, *request*, *msg=None*)
Failed to retrieve server log levels.

exception arango.exceptions.**ServerLogLevelSetError**(*resp*, *request*, *msg=None*)
Failed to set server log levels.

exception arango.exceptions.**ServerReadLogError**(*resp*, *request*, *msg=None*)
Failed to retrieve global log.

exception arango.exceptions.**ServerReloadRoutingError**(*resp*, *request*, *msg=None*)
Failed to reload routing details.

exception arango.exceptions.**ServerRequiredDBVersionError**(*resp*, *request*, *msg=None*)
Failed to retrieve server target version.

exception arango.exceptions.**ServerRoleError**(*resp*, *request*, *msg=None*)
Failed to retrieve server role in a cluster.

exception arango.exceptions.**ServerRunTestsError**(*resp*, *request*, *msg=None*)
Failed to execute server tests.

exception arango.exceptions.**ServerShutdownError**(*resp*, *request*, *msg=None*)
Failed to initiate shutdown sequence.

exception arango.exceptions.**ServerStatisticsError**(*resp*, *request*, *msg=None*)
Failed to retrieve server statistics.

exception arango.exceptions.**ServerStatusError**(*resp*, *request*, *msg=None*)
Failed to retrieve server status.

exception arango.exceptions.**ServerTimeError**(*resp*, *request*, *msg=None*)
Failed to retrieve server system time.

exception arango.exceptions.**ServerVersionError**(*resp*, *request*, *msg=None*)
Failed to retrieve server version.

exception arango.exceptions.**TaskCreateError**(*resp*, *request*, *msg=None*)
Failed to create server task.

exception arango.exceptions.**TaskDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete server task.

exception arango.exceptions.**TaskGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve server task details.

exception arango.exceptions.**TaskListError**(*resp*, *request*, *msg=None*)
Failed to retrieve server tasks.

exception arango.exceptions.**TransactionExecuteError**(*resp*, *request*, *msg=None*)
Failed to execute transaction API request

exception arango.exceptions.**TransactionJobResultError**(*msg*)
Failed to retrieve transaction job result.

exception arango.exceptions.**TransactionStateError**(*msg*)
The transaction object was in bad state.

exception arango.exceptions.**UserCreateError**(*resp*, *request*, *msg=None*)
Failed to create user.

exception arango.exceptions.**UserDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete user.

exception arango.exceptions.**UserGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve user details.

exception arango.exceptions.**UserListError**(*resp*, *request*, *msg=None*)
Failed to retrieve users.

exception arango.exceptions.**UserReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace user.

exception arango.exceptions.**UserUpdateError**(*resp*, *request*, *msg=None*)
Failed to update user.

exception arango.exceptions.**VertexCollectionCreateError**(*resp*, *request*, *msg=None*)
Failed to create vertex collection.

exception arango.exceptions.**VertexCollectionDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete vertex collection.

exception arango.exceptions.**VertexCollectionListError**(*resp*, *request*, *msg=None*)
Failed to retrieve vertex collections.

exception arango.exceptions.**ViewCreateError**(*resp*, *request*, *msg=None*)
Failed to create view.

exception arango.exceptions.**ViewDeleteError**(*resp*, *request*, *msg=None*)
Failed to delete view.

exception arango.exceptions.**ViewGetError**(*resp*, *request*, *msg=None*)
Failed to retrieve view details.

exception arango.exceptions.**ViewListError**(*resp*, *request*, *msg=None*)
Failed to retrieve views.

exception arango.exceptions.**ViewRenameError**(*resp*, *request*, *msg=None*)
Failed to rename view.

exception arango.exceptions.**ViewReplaceError**(*resp*, *request*, *msg=None*)
Failed to replace view.

exception arango.exceptions.**ViewUpdateError**(*resp*, *request*, *msg=None*)
Failed to update view.

exception arango.exceptions.**WALConfigureError**(*resp*, *request*, *msg=None*)
Failed to configure WAL properties.

exception arango.exceptions.**WALFlushError**(*resp*, *request*, *msg=None*)
Failed to flush WAL.

exception arango.exceptions.**WALPropertiesError**(*resp*, *request*, *msg=None*)
Failed to retrieve WAL properties.

exception arango.exceptions.**WALTransactionListError**(*resp*, *request*, *msg=None*)
Failed to retrieve running WAL transactions.

3.21 Logging

In order to see full HTTP request response details, you can modify logger settings for Requests library, which python-arango uses under the hood:

```
import requests
import logging

try:
    # For Python 3
    from http.client import HTTPConnection
except ImportError:
    # For Python 2
    from httplib import HTTPConnection
HTTPConnection.debuglevel = 1

logging.basicConfig()
logging.getLogger().setLevel(logging.DEBUG)
requests_log = logging.getLogger("requests.packages.urllib3")
requests_log.setLevel(logging.DEBUG)
requests_log.propagate = True
```

Note: If python-arango's default HTTP client is overridden with a custom one, the code snippet above may not work as expected.

Alternatively, if you want to use your own loggers, see *Using Custom HTTP Clients* for an example.

3.22 Using Custom HTTP Clients

Python-arango lets you use your own HTTP clients for sending API requests to ArangoDB server. The default implementation uses the requests library.

Your HTTP client must inherit *arango.http.HTTPClient* and implement its abstract method *arango.http.HTTPClient.send_request()*. The method must return valid (fully populated) instances of *arango.response.Response*.

For example, let's say your HTTP client needs:

- Automatic retries
- Additional HTTP header called `x-my-header`
- SSL certificate verification disabled
- Custom logging

Your `CustomHTTPClient` class might look something like this:

```
import logging

from requests.adapters import HTTPAdapter
from requests import Session

from arango.response import Response
from arango.http import HTTPClient


class CustomHTTPClient(HTTPClient):
    """My custom HTTP client with cool features."""
```

(continues on next page)

(continued from previous page)

```
    def __init__(self):
        self._session = Session()

        # Initialize your logger.
        self._logger = logging.getLogger('my_logger')

        # Add your request headers.
        self._session.headers.update({'x-my-header': 'true'})

        # Enable retries.
        adapter = HTTPAdapter(max_retries=5)
        self._session.mount('https://', adapter)

    def send_request(self,
                     method,
                     url,
                     params=None,
                     data=None,
                     headers=None,
                     auth=None):

        # Add your own debug statement.
        self._logger.debug('Sending request to {}'.format(url))

        # Send a request.
        response = self._session.request(
            method=method,
            url=url,
            params=params,
            data=data,
            headers=headers,
            auth=auth,
            verify=False  # Disable SSL verification
        )
        self._logger.debug('Got {}'.format(response.status_code))

        # Return an instance of arango.response.Response per spec.
        return Response(
            method=response.request.method,
            url=response.url,
            headers=response.headers,
            status_code=response.status_code,
            status_text=response.reason,
            raw_body=response.text,
        )
```

Then you would inject your client as follows:

```
from arango import ArangoClient

# from my_module import CustomHTTPClient

client = ArangoClient(
    protocol='http',
    host='localhost',
    port=8529,
    http_client=CustomHTTPClient()
```

(continues on next page)

(continued from previous page)

```
)
```

For more information on how to configure a `requests.Session` object, refer to requests documentation.

3.23 Contributing

3.23.1 Requirements

Before submitting a pull request on GitHub, please make sure you meet the following requirements:

- The pull request points to dev branch.
- Changes are squashed into a single commit. I like to use git rebase for this.
- Commit message is in present tense. For example, "Fix bug" is good while "Fixed bug" is not.
- Sphinx-compatible docstrings.
- PEP8 compliance.
- No missing docstrings or commented-out lines.
- Test coverage remains at %100. If a piece of code is trivial and does not need unit tests, use this to exclude it from coverage.
- No build failures on Travis CI. Builds automatically trigger on pull request submissions.
- Documentation is kept up-to-date with the new changes (see below).

Warning: The dev branch is occasionally rebased, and its commit history may be overwritten in the process. Before you begin your feature work, git fetch or pull to ensure that your local branch has not diverged. If you see git conflicts and want to start with a clean slate, run the following commands:

```
~$ git checkout dev
~$ git fetch origin
~$ git reset --hard origin/dev  # THIS WILL WIPE ALL LOCAL CHANGES
```

3.23.2 Style

To ensure PEP8 compliance, run flake8:

```
~$ pip install flake8
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ flake8
```

If there is a good reason to ignore a warning, see here on how to exclude it.

3.23.3 Testing

To test your changes, you can run the integration test suite that comes with **python-arango**. It uses pytest and requires an actual ArangoDB instance.

To run the test suite (use your own host, port and root password):

```
~$ pip install pytest
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ py.test --complete --host=127.0.0.1 --port=8529 --passwd=passwd
```

To run the test suite with coverage report:

```
~$ pip install coverage pytest pytest-cov
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango
~$ py.test --complete --host=127.0.0.1 --port=8529 --passwd=passwd --cov=kq
```

As the test suite creates real databases and jobs, it should only be run in development environments.

3.23.4 Documentation

The documentation including the README is written in reStructuredText and uses Sphinx. To build an HTML version on your local machine:

```
~$ pip install sphinx sphinx_rtd_theme
~$ git clone https://github.com/joowani/python-arango.git
~$ cd python-arango/docs
~$ sphinx-build . build  # Open build/index.html in a browser
```

As always, thank you for your contribution!

3.24 API Specification

This page contains the specification for all classes and methods available in python-arango.

3.24.1 ArangoClient

class arango.client.**ArangoClient**(*protocol=u'http'*, *host=u'127.0.0.1'*, *port=8529*, *http_client=None*)

ArangoDB client.

Parameters

- **protocol** (*str | unicode*) – Internet transfer protocol (default: "http").
- **host** (*str | unicode*) – ArangoDB host (default: "127.0.0.1").
- **port** (*int*) – ArangoDB port (default: 8529).
- **http_client** (arango.http.HTTPClient) – User-defined HTTP client.

base_url

Return the ArangoDB base URL.

Returns ArangoDB base URL.

Return type str | unicode

db(*name=u'_system'*, *username=u'root'*, *password=u''*, *verify=False*)

Connect to a database and return the database API wrapper.

Parameters

- **name** (*str | unicode*) – Database name.
- **username** (*str | unicode*) – Username for basic authentication.
- **password** (*str | unicode*) – Password for basic authentication.
- **verify** (*bool*) – Verify the connection by sending a test request.

Returns Standard database API wrapper.

Return type *arango.database.StandardDatabase*

Raises ***arango.exceptions.ServerConnectionError*** – If **verify** was set to True and the connection to ArangoDB fails.

host
Return the ArangoDB host.

Returns ArangoDB host.

Return type str | unicode

port
Return the ArangoDB port.

Returns ArangoDB port.

Return type int

protocol
Return the internet transfer protocol (e.g. "http").

Returns Internet transfer protocol.

Return type str | unicode

version
Return the client version.

Returns Client version.

Return type str | unicode

3.24.2 AsyncDatabase

class arango.database.**AsyncDatabase**(*connection*, *return_result*)
Database API wrapper tailored specifically for async execution.

See *arango.database.StandardDatabase.begin_async_execution()*.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.AsyncJob*, which you can use to retrieve results from server once available. If set to False, API executions return None and no results are stored on server.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status*, *count=None*)
Return IDs of async jobs with given status.

Parameters

- **status** (*str | unicode*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str | unicode]

Raises ***arango.exceptions.AsyncJobListError*** – If retrieval fails.

clear_async_jobs(*threshold=None*)
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

collection(*name*)
Return the standard collection API wrapper.

Parameters **name** (*str | unicode*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections()
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str | unicode

create_collection(*name*, *sync=False*, *compact=True*, *system=False*, *journal_size=None*, *edge=False*, *volatile=False*, *user_keys=True*, *key_increment=None*, *key_offset=None*, *key_generator=u'traditional'*, *shard_fields=None*, *shard_count=None*, *index_bucket_count=None*, *replication_factor=None*, *shard_like=None*, *sync_replication=None*, *enforce_replication_factor=None*)
Create a new collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **sync** (*bool*) – If set to True, document operations via the collection will block until synchronized to disk by default.

- **compact** (*bool*) – If set to True, the collection is compacted. Applies only to MMFiles storage engine.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **journal_size** (*int*) – Max size of the journal in bytes.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **volatile** (*bool*) – If set to True, collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data.
- **key_generator** (*str | unicode*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str | unicode]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **index_bucket_count** (*int*) – Number of buckets into which indexes using hash tables are split. The default is 16, and this number has to be a power of 2 and less than or equal to 1024. For large collections, one should increase this to avoid long pauses when the hash table has to be initially built or re-sized, since buckets are re-sized individually and can be initially built in parallel. For instance, 64 may be a sensible value for 100 million documents.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str | unicode*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name*, *users=None*)

Create a new database.

Parameters

- **name** (*str | unicode*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name*, *edge_definitions=None*, *orphan_collections=None*, *smart=None*, *smart_field=None*, *shard_count=None*)

Create a new graph.

Parameters

- **name** (*str | unicode*) – Graph name.
- **edge_definitions** (*[dict]*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str | unicode]*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | unicode*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name*, *command*, *params=None*, *period=None*, *offset=None*, *task_id=None*)
Create a new server task.

Parameters

- **name** (*str | unicode*) – Name of the server task.
- **command** (*str | unicode*) – Javascript command to execute.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **period** (*int*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int*) – Initial delay before execution in seconds.
- **task_id** (*str | unicode*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises *arango.exceptions.TaskCreateError* – If create fails.

create_user(*username*, *password*, *active=True*, *extra=None*)
Create a new user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – Password.
- **active** (*bool*) – True if user is active, False otherwise.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name*, *view_type*, *properties=None*)
Create a view.

Parameters

- **name** (*str | unicode*) – View name.
- **view_type** (*str | unicode*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases()
Return the names all databases.

Returns Database names.

Return type [str | unicode]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str | unicode

delete_collection(*name*, *ignore_missing=False*, *system=None*)
Delete the collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name*, *ignore_missing=False*)
Delete the database.

Parameters

- **name** (*str | unicode*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *return_old=False*, *sync=None*, *silent=False*)
Delete a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name*, *ignore_missing=False*, *drop_collections=None*)
Drop the graph of the given name from the database.

Parameters

- **name** (*str | unicode*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id*, *ignore_missing=False*)
Delete a server task.

Parameters

- **task_id** (*str | unicode*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username*, *ignore_missing=False*)
Delete a user.

Parameters

- **username** (*str | unicode*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.UserDeleteError*** – If delete fails.

delete_view(*name*, *ignore_missing=False*)
Delete a view.

Parameters **name** (*str | unicode*) – View name.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type dict

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details()
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document*, *rev=None*, *check_rev=True*)
Return a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- *arango.exceptions.DocumentGetError* – If retrieval fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

echo()
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises *arango.exceptions.ServerEchoError* – If retrieval fails.

endpoints()
Return coordinate endpoints. This method is for clusters only.

Returns List of endpoints.

Return type [str | unicode]

Raises *arango.exceptions.ServerEndpointsError* – If retrieval fails.

engine()
Return the database engine details.

Returns Database engine details.

Return type str | unicode

Raises *arango.exceptions.ServerEngineError* – If retrieval fails.

execute_transaction(*command*, *params=None*, *read=None*, *write=None*, *sync=None*, *timeout=None*, *max_size=None*, *allow_implicit=None*, *intermediate_commit_count=None*, *intermediate_commit_size=None*)

Execute raw Javascript command in transaction.

Parameters

- **command** (*str | unicode*) – Javascript command to execute.
- **read** (*[str | unicode]*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str | unicode]*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int*) – Max transaction size limit in bytes. Applies only to RocksDB storage engine.
- **allow_implicit** (*bool*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.
- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

Returns Return value of **command**.

Return type str | unicode

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx

Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name*)

Return the graph API wrapper.

Parameters **name** (*str | unicode*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs()

List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name*)
Check if collection exists in the database.

Parameters **name** (*str | unicode*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name*)
Check if a database exists.

Parameters **name** (*str | unicode*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document*, *rev=None*, *check_rev=True*)
Check if a document exists.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name*)
Check if a graph exists in the database.

Parameters **name** (*str | unicode*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username*)
Check if user exists.

Parameters **username** (*str | unicode*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection*, *document*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)
Insert a new document.

Parameters

- **collection** (*str | unicode*) – Collection name.

- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

log_levels()
Return current logging levels.

Returns Current logging levels.

Return type dict

name
Return database name.

Returns Database name.

Return type str | unicode

permission(*username*, *database*, *collection=None*)
Return user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns Permission for given database or collection.

Return type str | unicode

Raise arango.exceptions.PermissionGetError: If retrieval fails.

permissions(*username*)
Return user permissions for all databases and collections.

Parameters **username** (*str | unicode*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raise arango.exceptions.PermissionListError: If retrieval fails.

ping()
Ping the ArangoDB server by sending a test request.

Returns Response code from server.

Return type int

Raises ***arango.exceptions.ServerConnectionError*** – If ping fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties()
Return database properties.

Returns Database properties.

Return type dict

Raises ***arango.exceptions.DatabasePropertiesError*** – If retrieval fails.

read_log(*upto=None*, *level=None*, *start=None*, *size=None*, *offset=None*, *search=None*, *sort=None*)
Read the global log from server.

Parameters

- **upto** (*str | unicode | int*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*str | unicode | int*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str | unicode*) – Return only the log entries containing the given text.
- **sort** (*str | unicode*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises ***arango.exceptions.ServerReadLogError*** – If read fails.

reload_routing()
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises ***arango.exceptions.ServerReloadRoutingError*** – If reload fails.

rename_view(*name*, *new_name*)
Rename a view.

Parameters **name** (*str | unicode*) – View name.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewRenameError*** – If delete fails.

replace_document(*document*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)

Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username*, *password*, *active=None*, *extra=None*)

Replace a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name*, *properties*)

Replace a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewReplaceError* – If replace fails.

required_db_version()
Return required version of target database.

Returns Required version of target database.

Return type str | unicode

Raises *arango.exceptions.ServerRequiredDBVersionError* – If retrieval fails.

reset_permission(*username*, *database*, *collection=None*)
Reset user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role()
Return server role in cluster.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY" or "UNDEFINED".

Return type str | unicode

Raises *arango.exceptions.ServerRoleError* – If retrieval fails.

run_tests(*tests*)
Run available unittests on the server.

Parameters **tests** (*[str | unicode]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises *arango.exceptions.ServerRunTestsError* – If execution fails.

set_log_levels(***kwargs*)
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown()
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises *arango.exceptions.ServerShutdownError* – If shutdown fails.

statistics(*description=False*)
Return server statistics.

Returns Server statistics.

Return type dict

Raises *arango.exceptions.ServerStatisticsError* – If retrieval fails.

status()
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises *arango.exceptions.ServerStatusError* – If retrieval fails.

task(*task_id*)
Return the details of an active server task.

Parameters **task_id** (*str | unicode*) – Server task ID.

Returns Server task details.

Return type dict

Raises *arango.exceptions.TaskGetError* – If retrieval fails.

tasks()
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises *arango.exceptions.TaskListError* – If retrieval fails.

time()
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

update_document(*document*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username*, *permission*, *database*, *collection=None*)

Update user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.
- **permission** (*str | unicode*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username*, *password=None*, *active=None*, *extra=None*)

Update a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name*, *properties*)

Update a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username*)
Return user details.

Parameters **username** (*str | unicode*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str | unicode

users()
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version()
Return ArangoDB server version.

Returns Server version.

Return type str | unicode

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name*)
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views()
Return list of views.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.24.3 AsyncJob

class arango.job.**AsyncJob**(*connection*, *job_id*, *response_handler*)
Job for tracking and retrieving result of an async execution.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **job_id** (*str | unicode*) – Async job ID.
- **response_handler** (*callable*) – HTTP response handler.

cancel(*ignore_missing=False*)
Cancel the async job.

An async job cannot be cancelled once it is taken out of the queue.

Parameters **ignore_missing** (*bool*) – Do not raise an exception on missing job.

Returns True if job was cancelled successfully, False if the job was not found but **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.AsyncJobCancelError*** – If cancel fails.

clear(*ignore_missing=False*)
Delete the job result from the server.

Parameters **ignore_missing** (*bool*) – Do not raise an exception on missing job.

Returns True if result was deleted successfully, False if the job was not found but **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If delete fails.

id
Return the async job ID.

Returns Async job ID.

Return type str | unicode

result()
Return the async job result from server.

If the job raised an exception, it is propagated up at this point.

Once job result is retrieved, it is deleted from server and subsequent queries for result will fail.

Returns Async job result.

Return type str | unicode | bool | int | list | dict

Raises

- ***arango.exceptions.ArangoError*** – If the job raised an exception.
- ***arango.exceptions.AsyncJobResultError*** – If retrieval fails.

status()
Return the async job status from server.

Once a job result is retrieved via func:*arango.job.AsyncJob.result* method, it is deleted from server and subsequent status queries will fail.

Returns Async job status. Possible values are "pending" (job is still in queue), "done" (job finished or raised an error), or "cancelled" (job was cancelled before completion).

Return type str | unicode

Raises ***arango.exceptions.AsyncJobStatusError*** – If retrieval fails.

3.24.4 AQL

class arango.aql.**AQL**(*connection*, *executor*)
AQL (ArangoDB Query Language) API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.

cache
Return the query cache API wrapper.

Returns Query cache API wrapper.

Return type *arango.aql.AQLQueryCache*

clear_slow_queries()
Clear slow AQL queries.

Returns True if slow queries were cleared successfully.

Return type bool

Raises ***arango.exceptions.AQLQueryClearError*** – If operation fails.

create_function(*name*, *code*)
Create a new AQL function.

Parameters

- **name** (*str | unicode*) – AQL function name.
- **code** (*str | unicode*) – Function definition in Javascript.

Returns Whether the AQL function was newly created or an existing one was replaced.

Return type dict

Raises ***arango.exceptions.AQLFunctionCreateError*** – If create fails.

delete_function(*name*, *group=False*, *ignore_missing=False*)
Delete an AQL function.

Parameters

- **name** (*str | unicode*) – AQL function name.
- **group** (*bool*) – If set to True, value of parameter **name** is treated as a namespace prefix, and all functions in the namespace are deleted. If set to False, the value of **name** must be a fully qualified function name including any namespaces.

- **ignore_missing** (*bool*) – Do not raise an exception on missing function.

Returns Number of AQL functions deleted if operation was successful, False if function(s) was not found and **ignore_missing** was set to True.

Return type dict | bool

Raises *arango.exceptions.AQLFunctionDeleteError* – If delete fails.

execute(*query*, *count=False*, *batch_size=None*, *ttl=None*, *bind_vars=None*, *full_count=None*, *max_plans=None*, *optimizer_rules=None*, *cache=None*, *memory_limit=0*, *fail_on_warning=None*, *profile=None*, *max_transaction_size=None*, *max_warning_count=None*, *intermediate_commit_count=None*, *intermediate_commit_size=None*, *satellite_sync_wait=None*, *read_collections=None*, *write_collections=None*, *stream=None*, *skip_inaccessible_cols=None*)

Execute the query and return the result cursor.

Parameters

- **query** (*str | unicode*) – Query to execute.
- **count** (*bool*) – If set to True, the total document count is included in the result cursor.
- **batch_size** (*int*) – Number of documents fetched by the cursor in one round trip.
- **ttl** (*int*) – Server side time-to-live for the cursor in seconds.
- **bind_vars** (*dict*) – Bind variables for the query.
- **full_count** (*bool*) – This parameter applies only to queries with LIMIT clauses. If set to True, the number of matched documents before the last LIMIT clause executed is included in teh cursor. This is similar to MySQL SQL_CALC_FOUND_ROWS hint. Using this disables a few LIMIT optimizations and may lead to a longer query execution.
- **max_plans** (*int*) – Max number of plans the optimizer generates.
- **optimizer_rules** (*[str | unicode]*) – List of optimizer rules.
- **cache** (*bool*) – If set to True, the query cache is used. The operation mode of the query cache must be set to "on" or "demand".
- **memory_limit** (*int*) – Max amount of memory the query is allowed to use in bytes. If the query goes over the limit, it fails with error "resource limit exceeded". Value 0 indicates no limit.
- **fail_on_warning** (*bool*) – If set to True, the query throws an exception instead of producing a warning. This parameter can be used during development to catch issues early. If set to False, warnings are returned with the query result. There is a server configuration option "–query.fail-on-warning" for setting the default value for this behaviour so it does not need to be set per-query.
- **profile** (*bool*) – Return additional profiling details in the cursor, unless the query cache is used.
- **max_transaction_size** (*int*) – Transaction size limit in bytes. Applies only to RocksDB storage engine.
- **max_warning_count** (*int*) – Max number of warnings returned.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.
- **satellite_sync_wait** (*int | float*) – Number of seconds in which the server must synchronize the satellite collections involved in the query. When the threshold is reached, the query is stopped. Available only for enterprise version of ArangoDB.
- **read_collections** (*[str | unicode]*) – Names of collections read during query execution. Required for *transactions*.
- **write_collections** (*[str | unicode]*) – Names of collections written to during query execution. Required for *transactions*.
- **stream** (*bool*) – If set to True, query is executed in streaming fashion: query result is not stored server-side but calculated on the fly. Note: long-running queries hold collection locks for as long as the cursor exists. If set to False, query is executed right away in its entirety. Results are either returned right away (if the result set is small enough), or stored server-side and accessible via cursors (while respecting the ttl). You should use this parameter only for short-running queries or without exclusive locks (write-locks on MMFiles). Note: parameters **cache**, **count** and **full_count** do not work for streaming queries. Query statistics, warnings and profiling data are made available only after the query is finished. Default value is False.
- **skip_inaccessible_cols** (*bool*) – If set to True, collections without user access are skipped, and query executes normally instead of raising an error. This helps certain use cases: a graph may contain several collections, and users with different access levels may execute the same query. This parameter lets you limit the result set by user access. Cannot be used in *transactions* and is available only for enterprise version of ArangoDB. Default value is False.

Returns Result cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.AQLQueryExecuteError*** – If execute fails.

explain(*query*, *all_plans=False*, *max_plans=None*, *opt_rules=None*)

Inspect the query and return its metadata without executing it.

Parameters

- **query** (*str | unicode*) – Query to inspect.
- **all_plans** (*bool*) – If set to True, all possible execution plans are returned in the result. If set to False, only the optimal plan is returned.
- **max_plans** (*int*) – Total number of plans generated by the optimizer.
- **opt_rules** (*list*) – List of optimizer rules.

Returns Execution plan, or plans if **all_plans** was set to True.

Return type dict | list

Raises ***arango.exceptions.AQLQueryExplainError*** – If explain fails.

functions()

List the AQL functions defined in the database.

Returns AQL functions.

Return type [dict]

Raises *arango.exceptions.AQLFunctionListError* – If retrieval fails.

kill(*query_id*)
Kill a running query.

Parameters **query_id** (*str | unicode*) – Query ID.

Returns True if kill request was sent successfully.

Return type bool

Raises *arango.exceptions.AQLQueryKillError* – If the send fails.

queries()
Return the currently running AQL queries.

Returns Running AQL queries.

Return type [dict]

Raises *arango.exceptions.AQLQueryListError* – If retrieval fails.

set_tracking(*enabled=None*, *max_slow_queries=None*, *slow_query_threshold=None*, *max_query_string_length=None*, *track_bind_vars=None*, *track_slow_queries=None*)
Configure AQL query tracking properties

Returns Updated AQL query tracking properties.

Return type dict

Raises *arango.exceptions.AQLQueryTrackingSetError* – If operation fails.

slow_queries()
Return a list of all slow AQL queries.

Returns Slow AQL queries.

Return type [dict]

Raises *arango.exceptions.AQLQueryListError* – If retrieval fails.

tracking()
Return AQL query tracking properties.

Returns AQL query tracking properties.

Return type dict

Raises *arango.exceptions.AQLQueryTrackingGetError* – If retrieval fails.

validate(*query*)
Parse and validate the query without executing it.

Parameters **query** (*str | unicode*) – Query to validate.

Returns Query details.

Return type dict

Raises *arango.exceptions.AQLQueryValidateError* – If validation fails.

3.24.5 AQLQueryCache

class arango.aql.**AQLQueryCache**(*connection*, *executor*)
AQL Query Cache API wrapper.

clear()
Clear the query cache.

Returns True if query cache was cleared successfully.

Return type dict

Raises ***arango.exceptions.AQLCacheClearError*** – If operation fails.

configure(*mode=None*, *limit=None*)
Configure the query cache properties.

Parameters

- **mode** (*str | unicode*) – Operation mode. Allowed values are "off", "on" and "demand".
- **limit** (*int*) – Max number of query results to be stored.

Returns Query cache properties.

Return type dict

Raises ***arango.exceptions.AQLCacheConfigureError*** – If operation fails.

entries()
Return the query cache entries.

Returns Query cache entries.

Return type list

Raises ***AQLCacheEntriesError*** – If retrieval fails.

properties()
Return the query cache properties.

Returns Query cache properties.

Return type dict

Raises ***arango.exceptions.AQLCachePropertiesError*** – If retrieval fails.

3.24.6 BatchDatabase

class arango.database.**BatchDatabase**(*connection*, *return_result*)
Database API wrapper tailored specifically for batch execution.

See *arango.database.StandardDatabase.begin_batch_execution()*.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.BatchJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status*, *count=None*)
Return IDs of async jobs with given status.

Parameters

- **status** (*str | unicode*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str | unicode]

Raises ***arango.exceptions.AsyncJobListError*** – If retrieval fails.

clear_async_jobs(*threshold=None*)
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

collection(*name*)
Return the standard collection API wrapper.

Parameters **name** (*str | unicode*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections()
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

commit()
Execute the queued requests in a single batch API request.

If **return_result** parameter was set to True during initialization, *arango.job.BatchJob* instances are populated with results.

Returns Batch jobs, or None if **return_result** parameter was set to False during initialization.

Return type [*arango.job.BatchJob*] | None

Raises

- ***arango.exceptions.BatchStateError*** – If batch state is invalid (e.g. batch was already committed or the response size did not match expected).
- ***arango.exceptions.BatchExecuteError*** – If commit fails.

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str | unicode

create_collection(*name*, *sync=False*, *compact=True*, *system=False*, *journal_size=None*, *edge=False*, *volatile=False*, *user_keys=True*, *key_increment=None*, *key_offset=None*, *key_generator=u'traditional'*, *shard_fields=None*, *shard_count=None*, *index_bucket_count=None*, *replication_factor=None*, *shard_like=None*, *sync_replication=None*, *enforce_replication_factor=None*)

Create a new collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **sync** (*bool*) – If set to True, document operations via the collection will block until synchronized to disk by default.
- **compact** (*bool*) – If set to True, the collection is compacted. Applies only to MMFiles storage engine.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **journal_size** (*int*) – Max size of the journal in bytes.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **volatile** (*bool*) – If set to True, collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data.
- **key_generator** (*str | unicode*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str | unicode]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **index_bucket_count** (*int*) – Number of buckets into which indexes using hash tables are split. The default is 16, and this number has to be a power of 2 and less than or equal to 1024. For large collections, one should increase this to avoid long pauses when the hash table has to be initially built or re-sized, since buckets are re-sized individually and can be initially built in parallel. For instance, 64 may be a sensible value for 100 million documents.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).

- **shard_like** (*str | unicode*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name*, *users=None*)

Create a new database.

Parameters

- **name** (*str | unicode*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name*, *edge_definitions=None*, *orphan_collections=None*, *smart=None*, *smart_field=None*, *shard_count=None*)

Create a new graph.

Parameters

- **name** (*str | unicode*) – Graph name.
- **edge_definitions** (*[dict]*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str | unicode]*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | unicode*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.

- **shard_count** (*int*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name*, *command*, *params=None*, *period=None*, *offset=None*, *task_id=None*)

Create a new server task.

Parameters

- **name** (*str | unicode*) – Name of the server task.
- **command** (*str | unicode*) – Javascript command to execute.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **period** (*int*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int*) – Initial delay before execution in seconds.
- **task_id** (*str | unicode*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises ***arango.exceptions.TaskCreateError*** – If create fails.

create_user(*username*, *password*, *active=True*, *extra=None*)

Create a new user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – Password.
- **active** (*bool*) – True if user is active, False otherwise.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserCreateError*** – If create fails.

create_view(*name*, *view_type*, *properties=None*)

Create a view.

Parameters

- **name** (*str | unicode*) – View name.
- **view_type** (*str | unicode*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases()
Return the names all databases.

Returns Database names.

Return type [str | unicode]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str | unicode

delete_collection(*name*, *ignore_missing=False*, *system=None*)
Delete the collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name*, *ignore_missing=False*)
Delete the database.

Parameters

- **name** (*str | unicode*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *return_old=False*, *sync=None*, *silent=False*)
Delete a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name*, *ignore_missing=False*, *drop_collections=None*)

Drop the graph of the given name from the database.

Parameters

- **name** (*str | unicode*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id*, *ignore_missing=False*)

Delete a server task.

Parameters

- **task_id** (*str | unicode*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username*, *ignore_missing=False*)
Delete a user.

Parameters

- **username** (*str | unicode*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.UserDeleteError* – If delete fails.

delete_view(*name*, *ignore_missing=False*)
Delete a view.

Parameters **name** (*str | unicode*) – View name.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type dict

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details()
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document*, *rev=None*, *check_rev=True*)
Return a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- *arango.exceptions.DocumentGetError* – If retrieval fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

echo()
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises *arango.exceptions.ServerEchoError* – If retrieval fails.

endpoints()
Return coordinate endpoints. This method is for clusters only.

Returns List of endpoints.

Return type [str | unicode]

Raises ***arango.exceptions.ServerEndpointsError*** – If retrieval fails.

engine()
Return the database engine details.

Returns Database engine details.

Return type str | unicode

Raises ***arango.exceptions.ServerEngineError*** – If retrieval fails.

execute_transaction(*command*, *params=None*, *read=None*, *write=None*, *sync=None*, *timeout=None*, *max_size=None*, *allow_implicit=None*, *intermediate_commit_count=None*, *intermediate_commit_size=None*)
Execute raw Javascript command in transaction.

Parameters

- **command** (*str | unicode*) – Javascript command to execute.
- **read** (*[str | unicode]*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str | unicode]*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int*) – Max transaction size limit in bytes. Applies only to RocksDB storage engine.
- **allow_implicit** (*bool*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.
- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

Returns Return value of **command**.

Return type str | unicode

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name*)
Return the graph API wrapper.

Parameters **name** (*str | unicode*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs()
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name*)
Check if collection exists in the database.

Parameters **name** (*str | unicode*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name*)
Check if a database exists.

Parameters **name** (*str | unicode*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document*, *rev=None*, *check_rev=True*)
Check if a document exists.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name*)
Check if a graph exists in the database.

Parameters **name** (*str | unicode*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username*)
Check if user exists.

Parameters **username** (*str | unicode*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection*, *document*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)
Insert a new document.

Parameters

- **collection** (*str | unicode*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

log_levels()
Return current logging levels.

Returns Current logging levels.

Return type dict

name
Return database name.

Returns Database name.

Return type str | unicode

permission(*username*, *database*, *collection=None*)
Return user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns Permission for given database or collection.

Return type str | unicode

Raise arango.exceptions.PermissionGetError: If retrieval fails.

permissions(*username*)
Return user permissions for all databases and collections.

Parameters **username** (*str | unicode*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raise arango.exceptions.PermissionListError: If retrieval fails.

ping()
Ping the ArangoDB server by sending a test request.

Returns Response code from server.

Return type int

Raises ***arango.exceptions.ServerConnectionError*** – If ping fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties()
Return database properties.

Returns Database properties.

Return type dict

Raises ***arango.exceptions.DatabasePropertiesError*** – If retrieval fails.

queued_jobs()
Return the queued batch jobs.

Returns Queued batch jobs or None if **return_result** parameter was set to False during initialization.

Return type [*arango.job.BatchJob*] | None

read_log(*upto=None*, *level=None*, *start=None*, *size=None*, *offset=None*, *search=None*, *sort=None*)
Read the global log from server.

Parameters

- **upto** (*str | unicode | int*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*str | unicode | int*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).

- **search** (*str | unicode*) – Return only the log entries containing the given text.
- **sort** (*str | unicode*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises ***arango.exceptions.ServerReadLogError*** – If read fails.

reload_routing()
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises ***arango.exceptions.ServerReloadRoutingError*** – If reload fails.

rename_view(*name*, *new_name*)
Rename a view.

Parameters **name** (*str | unicode*) – View name.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewRenameError*** – If delete fails.

replace_document(*document*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username*, *password*, *active=None*, *extra=None*)
Replace a user.

Parameters

- **username** (*str | unicode*) – Username.

- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserReplaceError* – If replace fails.

replace_view(*name*, *properties*)
Replace a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewReplaceError* – If replace fails.

required_db_version()
Return required version of target database.

Returns Required version of target database.

Return type str | unicode

Raises *arango.exceptions.ServerRequiredDBVersionError* – If retrieval fails.

reset_permission(*username*, *database*, *collection=None*)
Reset user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role()
Return server role in cluster.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY" or "UNDEFINED".

Return type str | unicode

Raises *arango.exceptions.ServerRoleError* – If retrieval fails.

run_tests(*tests*)
Run available unittests on the server.

Parameters **tests** (*[str | unicode]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises *arango.exceptions.ServerRunTestsError* – If execution fails.

set_log_levels(***kwargs*)
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown()
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises *arango.exceptions.ServerShutdownError* – If shutdown fails.

statistics(*description=False*)
Return server statistics.

Returns Server statistics.

Return type dict

Raises *arango.exceptions.ServerStatisticsError* – If retrieval fails.

status()
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises *arango.exceptions.ServerStatusError* – If retrieval fails.

task(*task_id*)
Return the details of an active server task.

Parameters **task_id** (*str | unicode*) – Server task ID.

Returns Server task details.

Return type dict

Raises *arango.exceptions.TaskGetError* – If retrieval fails.

tasks()
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises *arango.exceptions.TaskListError* – If retrieval fails.

time()
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

update_document(*document*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- *arango.exceptions.DocumentUpdateError* – If update fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

update_permission(*username*, *permission*, *database*, *collection=None*)
Update user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.
- **permission** (*str | unicode*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).

Returns True if access was granted successfully.

Return type bool

Raises *arango.exceptions.PermissionUpdateError* – If update fails.

update_user(*username*, *password=None*, *active=None*, *extra=None*)
Update a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name*, *properties*)
Update a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewUpdateError*** – If update fails.

user(*username*)
Return user details.

Parameters **username** (*str | unicode*) – Username.

Returns User details.

Return type dict

Raises ***arango.exceptions.UserGetError*** – If retrieval fails.

username
Return the username.

Returns Username.

Return type str | unicode

users()
Return all user details.

Returns List of user details.

Return type [dict]

Raises ***arango.exceptions.UserListError*** – If retrieval fails.

version()
Return ArangoDB server version.

Returns Server version.

Return type str | unicode

Raises ***arango.exceptions.ServerVersionError*** – If retrieval fails.

view(*name*)
Return view details.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewGetError*** – If retrieval fails.

views()
Return list of views.

Returns List of views.

Return type [dict]

Raises ***arango.exceptions.ViewListError*** – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.24.7 BatchJob

class arango.job.**BatchJob**(*response_handler*)
Job for tracking and retrieving result of batch execution.

Parameters **response_handler** (*callable*) – HTTP response handler.

id
Return the batch job ID.

Returns Batch job ID.

Return type str | unicode

result()
Return the batch job result.

If the job raised an exception, it is propagated up at this point.

Returns Batch job result.

Return type str | unicode | bool | int | list | dict

Raises

- ***arango.exceptions.ArangoError*** – If the job raised an exception.
- ***arango.exceptions.BatchJobResultError*** – If job result is not available (i.e. batch is not committed yet).

status()
Return the batch job status.

Returns Batch job status. Possible values are "pending" (job is still waiting for batch to be committed), or "done" (batch was committed and the job is updated with the result).

Return type str | unicode

3.24.8 Cursor

class arango.cursor.**Cursor**(*connection*, *init_data*, *cursor_type=u'cursor'*)

Cursor API wrapper.

Cursors fetch query results from ArangoDB server in batches. Cursor objects are *stateful* as they store the fetched items in-memory. They must not be shared across threads without proper locking mechanism.

In transactions, the entire result set is loaded into the cursor. Therefore you must be mindful of client-side memory capacity when running queries that can potentially return a large result set.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **init_data** (*dict | list*) – Cursor initialization data.
- **cursor_type** (*str | unicode*) – Cursor type ("cursor" or "export").

batch()

Return the current batch of results.

Returns Current batch.

Return type collections.deque

cached()

Return True if results are cached.

Returns True if results are cached.

Return type bool

close(*ignore_missing=False*)

Close the cursor and free any server resources tied to it.

Parameters **ignore_missing** (*bool*) – Do not raise exception on missing cursors.

Returns True if cursor was closed successfully, False if cursor was missing on the server and **ignore_missing** was set to True, None if there are no cursors to close server-side (e.g. result set is smaller than the batch size, or in transactions).

Return type bool | None

Raises

- ***arango.exceptions.CursorCloseError*** – If operation fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

count()

Return the total number of documents in the entire result set.

Returns Total number of documents, or None if the count option was not enabled during cursor initialization.

Return type int | None

empty()

Check if the current batch is empty.

Returns True if current batch is empty, False otherwise.

Return type bool

fetch()

Fetch the next batch from server and update the cursor.

Returns New batch details.

Return type dict

Raises

- ***arango.exceptions.CursorNextError*** – If batch retrieval fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

has_more()
Return True if more results are available on the server.

Returns True if more results are available on the server.

Return type bool

id
Return the cursor ID.

Returns Cursor ID.

Return type str | unicode

next()
Pop the next item from the current batch.

If current batch is empty/depleted, an API request is automatically sent to ArangoDB server to fetch the next batch and update the cursor.

Returns Next item in current batch.

Return type str | unicode | bool | int | list | dict

Raises

- **StopIteration** – If the result set is depleted.
- ***arango.exceptions.CursorNextError*** – If batch retrieval fails.
- ***arango.exceptions.CursorStateError*** – If cursor ID is not set.

pop()
Pop the next item from current batch.

If current batch is empty/depleted, an exception is raised. You must call *arango.cursor.Cursor.fetch()* to manually fetch the next batch from server.

Returns Next item in current batch.

Return type str | unicode | bool | int | list | dict

Raises ***arango.exceptions.CursorEmptyError*** – If current batch is empty.

profile()
Return cursor performance profile.

Returns Cursor performance profile.

Return type dict

statistics()
Return cursor statistics.

Returns Cursor statistics.

Return type dict

type
Return the cursor type.

Returns Cursor type ("cursor" or "export").

Return type str | unicode

warnings()
Return any warnings from the query execution.

Returns Warnings, or None if there are none.

Return type list

3.24.9 DefaultHTTPClient

class arango.http.**DefaultHTTPClient**
Default HTTP client implementation.

send_request(*method*, *url*, *params=None*, *data=None*, *headers=None*, *auth=None*)
Send an HTTP request.

Parameters

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **url** (*str | unicode*) – Request URL.
- **headers** (*dict*) – Request headers.
- **params** (*dict*) – URL (query) parameters.
- **data** (*str | unicode | bool | int | list | dict*) – Request payload.
- **auth** (*tuple*) – Username and password.

Returns HTTP response.

Return type *arango.response.Response*

3.24.10 StandardCollection

class arango.collection.**StandardCollection**(*connection*, *executor*, *name*)
Standard ArangoDB collection API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.
- **name** (*str | unicode*) – Collection name.

add_fulltext_index(*fields*, *min_length=None*)
Create a new fulltext index.

Parameters

- **fields** (*[str | unicode]*) – Document fields to index.
- **min_length** (*int*) – Minimum number of characters to index.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_geo_index(*fields*, *ordered=None*)
Create a new geo-spatial index.

Parameters

- **fields** (*str | unicode | list*) – A single document field or a list of document fields. If a single field is given, the field must have values that are lists with at least two floats. Documents with missing fields or invalid values are excluded.
- **ordered** (*bool*) – Whether the order is longitude, then latitude.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_hash_index(*fields*, *unique=None*, *sparse=None*, *deduplicate=None*)
Create a new hash index.

Parameters

- **fields** (*[str | unicode]*) – Document fields to index.
- **unique** (*bool*) – Whether the index is unique.
- **sparse** (*bool*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_persistent_index(*fields*, *unique=None*, *sparse=None*)
Create a new persistent index.

Unique persistent indexes on non-sharded keys are not supported in a cluster.

Parameters

- **fields** (*[str | unicode]*) – Document fields to index.
- **unique** (*bool*) – Whether the index is unique.
- **sparse** (*bool*) – Exclude documents that do not contain at least one of the indexed fields, or documents that have a value of None in any of the indexed fields.

Returns New index details.

Return type dict

Raises *arango.exceptions.IndexCreateError* – If create fails.

add_skiplist_index(*fields*, *unique=None*, *sparse=None*, *deduplicate=None*)
Create a new skiplist index.

Parameters

- **fields** (*[str | unicode]*) – Document fields to index.
- **unique** (*bool*) – Whether the index is unique.

- **sparse** (*bool*) – If set to True, documents with None in the field are also indexed. If set to False, they are skipped.
- **deduplicate** (*bool*) – If set to True, inserting duplicate index values from the same document triggers unique constraint errors.

Returns New index details.

Return type dict

Raises ***arango.exceptions.IndexCreateError*** – If create fails.

all (*skip=None*, *limit=None*)
Return all documents in the collection.

Parameters

- **skip** (*int*) – Number of documents to skip.
- **limit** (*int*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentGetError*** – If retrieval fails.

checksum (*with_rev=False*, *with_data=False*)
Return collection checksum.

Parameters

- **with_rev** (*bool*) – Include document revisions in checksum calculation.
- **with_data** (*bool*) – Include document data in checksum calculation.

Returns Collection checksum.

Return type str | unicode

Raises ***arango.exceptions.CollectionChecksumError*** – If retrieval fails.

configure (*sync=None*, *journal_size=None*)
Configure collection properties.

Parameters

- **sync** (*bool*) – Block until operations are synchronized to disk.
- **journal_size** (*int*) – Journal size in bytes.

Returns New collection properties.

Return type dict

Raises ***arango.exceptions.CollectionConfigureError*** – If operation fails.

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str | unicode

count ()
Return the total document count.

Returns Total document count.

Return type int

Raises *arango.exceptions.DocumentCountError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str | unicode

delete(*document*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *return_old=False*, *sync=None*, *silent=False*)
Delete a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- *arango.exceptions.DocumentDeleteError* – If delete fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

delete_index(*index_id*, *ignore_missing=False*)
Delete an index.

Parameters

- **index_id** (*str | unicode*) – Index ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing index.

Returns True if index was deleted successfully, False if index was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.IndexDeleteError* – If delete fails.

delete_many(*documents*, *return_old=False*, *check_rev=True*, *sync=None*, *silent=False*)
Delete multiple documents.

If deleting a document fails, the exception object is placed in the result list instead of document metadata.

Parameters

- **documents** (*[str | unicode | dict]*) – Document IDs, keys or bodies. Document bodies must contain the "_id" or "_key" fields.
- **return_old** (*bool*) – Include bodies of the old documents in the result.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises *arango.exceptions.DocumentDeleteError* – If delete fails.

delete_match (*filters*, *limit=None*, *sync=None*)
Delete matching documents.

Parameters

- **filters** (*dict*) – Document filters.
- **limit** (*int*) – Max number of documents to delete. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns Number of documents deleted.

Return type dict

Raises *arango.exceptions.DocumentDeleteError* – If delete fails.

export (*limit=None*, *count=False*, *batch_size=None*, *flush=False*, *flush_wait=None*, *ttl=None*, *filter_fields=None*, *filter_type=u'include'*)
Export all documents in the collection using a server cursor.

Parameters

- **flush** (*bool*) – If set to True, flush the write-ahead log prior to the export. If set to False, documents in the write-ahead log during the export are not included in the result.
- **flush_wait** (*int*) – Max wait time in seconds for write-ahead log flush.
- **count** (*bool*) – Include the document count in the server cursor.
- **batch_size** (*int*) – Max number of documents in the batch fetched by the cursor in one round trip.
- **limit** (*int*) – Max number of documents fetched by the cursor.
- **ttl** (*int*) – Time-to-live for the cursor on the server.
- **filter_fields** (*[str | unicode]*) – Document fields to filter with.
- **filter_type** (*str | unicode*) – Allowed values are "include" or "exclude".

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises *arango.exceptions.DocumentGetError* – If export fails.

find(*filters*, *skip=None*, *limit=None*)
Return all documents that match the given filters.

Parameters

- **filters** (*dict*) – Document filters.
- **skip** (*int*) – Number of documents to skip.
- **limit** (*int*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_by_text(*field*, *query*, *limit=None*)
Return documents that match the given fulltext query.

Parameters

- **field** (*str | unicode*) – Document field with fulltext index.
- **query** (*str | unicode*) – Fulltext query.
- **limit** (*int*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_box(*latitude1*, *longitude1*, *latitude2*, *longitude2*, *skip=None*, *limit=None*, *index=None*)
Return all documents in an rectangular area.

Parameters

- **latitude1** (*int | float*) – First latitude.
- **longitude1** (*int | float*) – First longitude.
- **latitude2** (*int | float*) – Second latitude.
- **longitude2** (*int | float*) – Second longitude
- **skip** (*int*) – Number of documents to skip.
- **limit** (*int*) – Max number of documents returned.
- **index** (*str | unicode*) – ID of the geo index to use (without the collection prefix). This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_radius(*latitude*, *longitude*, *radius*, *distance_field=None*)
Return documents within a given radius around a coordinate.

A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.

- **radius** (*int | float*) – Max radius.
- **distance_field** (*str | unicode*) – Document field used to indicate the distance to the given coordinate. This parameter is ignored in transactions.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_in_range (*field*, *lower*, *upper*, *skip=None*, *limit=None*)
Return documents within a given range in a random order.

A skiplist index must be defined in the collection to use this method.

Parameters

- **field** (*str | unicode*) – Document field name.
- **lower** (*int*) – Lower bound (inclusive).
- **upper** (*int*) – Upper bound (exclusive).
- **skip** (*int*) – Number of documents to skip.
- **limit** (*int*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

find_near (*latitude*, *longitude*, *limit=None*)
Return documents near a given coordinate.

Documents returned are sorted according to distance, with the nearest document being the first. If there are documents of equal distance, they are randomly chosen from the set until the limit is reached. A geo index must be defined in the collection to use this method.

Parameters

- **latitude** (*int | float*) – Latitude.
- **longitude** (*int | float*) – Longitude.
- **limit** (*int*) – Max number of documents returned.

Returns Document cursor.

Return type *arango.cursor.Cursor*

Raises **arango.exceptions.DocumentGetError** – If retrieval fails.

get (*document*, *rev=None*, *check_rev=True*)
Return a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

get_many(*documents*)
Return multiple documents ignoring any missing ones.

Parameters **documents** (*[str | unicode | dict]*) – List of document keys, IDs or bodies. Document bodies must contain the "_id" or "_key" fields.

Returns Documents. Missing ones are not included.

Return type [dict]

Raises ***arango.exceptions.DocumentGetError*** – If retrieval fails.

has(*document*, *rev=None*, *check_rev=True*)
Check if a document exists in the collection.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

ids()
Return the IDs of all documents in the collection.

Returns Document ID cursor.

Return type *arango.cursor.Cursor*

Raises ***arango.exceptions.DocumentIDsError*** – If retrieval fails.

import_bulk(*documents*, *halt_on_error=True*, *details=True*, *from_prefix=None*, *to_prefix=None*, *overwrite=None*, *on_duplicate=None*, *sync=None*)
Insert multiple documents into the collection.

This is faster than `arango.collection.Collection.insert_many()` but does not return as much information.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.

- **halt_on_error** (*bool*) – Halt the entire import on an error.
- **details** (*bool*) – If set to True, the returned result will include an additional list of detailed error messages.
- **from_prefix** (*str | unicode*) – String prefix prepended to the value of "_from" field in each edge document inserted. For example, prefix "foo" prepended to "_from": "bar" will result in "_from": "foo/bar". Applies only to edge collections.
- **to_prefix** (*str | unicode*) – String prefix prepended to the value of "_to" field in edge document inserted. For example, prefix "foo" prepended to "_to": "bar" will result in "_to": "foo/bar". Applies only to edge collections.
- **overwrite** (*bool*) – If set to True, all existing documents are removed prior to the import. Indexes are still preserved.
- **on_duplicate** (*str | unicode*) – Action to take on unique key constraint violations (for documents with "_key" fields). Allowed values are "error" (do not import the new documents and count them as errors), "update" (update the existing documents while preserving any fields missing in the new ones), "replace" (replace the existing documents with new ones), and "ignore" (do not import the new documents and count them as ignored, as opposed to counting them as errors). Options "update" and "replace" may fail on secondary unique key constraint violations.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns Result of the bulk import.

Return type dict

Raises ***arango.exceptions.DocumentInsertError*** – If import fails.

indexes()

Return the collection indexes.

Returns Collection indexes.

Return type [dict]

Raises ***arango.exceptions.IndexListError*** – If retrieval fails.

insert(*document*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)

Insert a new document.

Parameters

- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

insert_many(*documents*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)

Insert multiple documents.

If inserting a document fails, the exception object is placed in the result list instead of document metadata.

Parameters

- **documents** (*[dict]*) – List of new documents to insert. If they contain the "_key" or "_id" fields, the values are used as the keys of the new documents (auto-generated otherwise). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include bodies of the new documents in the returned metadata. Ignored if parameter **silent** is set to True
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate keys and the existing documents are replaced.
- **return_old** (*bool*) – Include body of the old documents if replaced. Applies only when value of **overwrite** is set to True.

Returns List of document metadata (e.g. document keys, revisions) and any exception, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

keys()

Return the keys of all documents in the collection.

Returns Document key cursor.

Return type *arango.cursor.Cursor*

Raises *arango.exceptions.DocumentKeysError* – If retrieval fails.

load()

Load the collection into memory.

Returns True if collection was loaded successfully.

Return type bool

Raises *arango.exceptions.CollectionLoadError* – If operation fails.

load_indexes()

Cache all indexes in the collection into memory.

Returns True if index was loaded successfully.

Return type bool

Raises *arango.exceptions.IndexLoadError* – If operation fails.

name
Return collection name.

Returns Collection name.

Return type str | unicode

properties()
Return collection properties.

Returns Collection properties.

Return type dict

Raises *arango.exceptions.CollectionPropertiesError* – If retrieval fails.

random()
Return a random document from the collection.

Returns A random document.

Return type dict

Raises *arango.exceptions.DocumentGetError* – If retrieval fails.

rename(*new_name*)
Rename the collection.

Renames may not be reflected immediately in async execution, batch execution or transactions. It is recommended to initialize new API wrappers after a rename.

Parameters **new_name** (*str | unicode*) – New collection name.

Returns True if collection was renamed successfully.

Return type bool

Raises *arango.exceptions.CollectionRenameError* – If rename fails.

replace(*document*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" or "_key" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- *arango.exceptions.DocumentReplaceError* – If replace fails.

- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

replace_many(*documents*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)

Replace multiple documents.

If replacing a document fails, the exception object is placed in the result list instead of document metadata.

Parameters

- **documents** (*[dict]*) – New documents to replace the old ones with. They must contain the "_id" or "_key" fields. Edge documents must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **return_new** (*bool*) – Include bodies of the new documents in the result.
- **return_old** (*bool*) – Include bodies of the old documents in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises *arango.exceptions.DocumentReplaceError* – If replace fails.

replace_match(*filters*, *body*, *limit=None*, *sync=None*)

Replace matching documents.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – New document body.
- **limit** (*int*) – Max number of documents to replace. If the limit is lower than the number of matched documents, random documents are chosen.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns Number of documents replaced.

Return type int

Raises *arango.exceptions.DocumentReplaceError* – If replace fails.

revision()

Return collection revision.

Returns Collection revision.

Return type str | unicode

Raises *arango.exceptions.CollectionRevisionError* – If retrieval fails.

rotate()

Rotate the collection journal.

Returns True if collection journal was rotated successfully.

Return type bool

Raises *arango.exceptions.CollectionRotateJournalError* – If rotate fails.

statistics()
Return collection statistics.

Returns Collection statistics.

Return type dict

Raises *arango.exceptions.CollectionStatisticsError* – If retrieval fails.

truncate()
Delete all documents in the collection.

Returns True if collection was truncated successfully.

Return type dict

Raises *arango.exceptions.CollectionTruncateError* – If operation fails.

unload()
Unload the collection from memory.

Returns True if collection was unloaded successfully.

Return type bool

Raises *arango.exceptions.CollectionUnloadError* – If operation fails.

update(*document*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" or "_key" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- *arango.exceptions.DocumentUpdateError* – If update fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

update_many(*documents*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)

Update multiple documents.

If updating a document fails, the exception object is placed in the result list instead of document metadata.

Parameters

- **documents** (*[dict]*) – Partial or full documents with the updated values. They must contain the "_id" or "_key" fields.
- **check_rev** (*bool*) – If set to True, revisions of **documents** (if given) are compared against the revisions of target documents.
- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include bodies of the new documents in the result.
- **return_old** (*bool*) – Include bodies of the old documents in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns List of document metadata (e.g. document keys, revisions) and any exceptions, or True if parameter **silent** was set to True.

Return type [dict | ArangoError] | bool

Raises ***arango.exceptions.DocumentUpdateError*** – If update fails.

update_match(*filters*, *body*, *limit=None*, *keep_none=True*, *sync=None*, *merge=True*)

Update matching documents.

Parameters

- **filters** (*dict*) – Document filters.
- **body** (*dict*) – Full or partial document body with the updates.
- **limit** (*int*) – Max number of documents to update. If the limit is lower than the number of matched documents, random documents are chosen. This parameter is not supported on sharded collections.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new ones overwriting the old ones.

Returns Number of documents updated.

Return type int

Raises ***arango.exceptions.DocumentUpdateError*** – If update fails.

username

Return the username.

Returns Username.

Return type str | unicode

3.24.11 StandardDatabase

class arango.database.**StandardDatabase**(*connection*)

Standard database API wrapper.

Parameters **connection** (*arango.connection.Connection*) – HTTP connection.

aql

Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status*, *count=None*)

Return IDs of async jobs with given status.

Parameters

- **status** (*str | unicode*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str | unicode]

Raises **arango.exceptions.AsyncJobListError** – If retrieval fails.

begin_async_execution(*return_result=True*)

Begin async execution.

Parameters **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.AsyncJob*, which you can use to retrieve results from server once available. If set to False, API executions return None and no results are stored on server.

Returns Database API wrapper built specifically for async execution.

Return type *arango.database.AsyncDatabase*

begin_batch_execution(*return_result=True*)

Begin batch execution.

Parameters **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.BatchJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.

Returns Database API wrapper built specifically for batch execution.

Return type *arango.database.BatchDatabase*

begin_transaction(*return_result=True*, *timeout=None*, *sync=None*, *read=None*, *write=None*)

Begin transaction.

Parameters

- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.TransactionJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.
- **read** (*[str | unicode]*) – Names of collections read during transaction. If not specified, they are added automatically as jobs are queued.

- **write** (*[str | unicode]*) – Names of collections written to during transaction. If not specified, they are added automatically as jobs are queued.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **sync** (*bool*) – Block until the transaction is synchronized to disk.

Returns Database API wrapper built specifically for transactions.

Return type *arango.database.TransactionDatabase*

clear_async_jobs(*threshold=None*)
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

collection(*name*)
Return the standard collection API wrapper.

Parameters **name** (*str | unicode*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections()
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str | unicode

create_collection(*name*, *sync=False*, *compact=True*, *system=False*, *journal_size=None*, *edge=False*, *volatile=False*, *user_keys=True*, *key_increment=None*, *key_offset=None*, *key_generator=u'traditional'*, *shard_fields=None*, *shard_count=None*, *index_bucket_count=None*, *replication_factor=None*, *shard_like=None*, *sync_replication=None*, *enforce_replication_factor=None*)
Create a new collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **sync** (*bool*) – If set to True, document operations via the collection will block until synchronized to disk by default.

- **compact** (*bool*) – If set to True, the collection is compacted. Applies only to MMFiles storage engine.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **journal_size** (*int*) – Max size of the journal in bytes.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **volatile** (*bool*) – If set to True, collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data.
- **key_generator** (*str | unicode*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".
- **shard_fields** (*[str | unicode]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **index_bucket_count** (*int*) – Number of buckets into which indexes using hash tables are split. The default is 16, and this number has to be a power of 2 and less than or equal to 1024. For large collections, one should increase this to avoid long pauses when the hash table has to be initially built or re-sized, since buckets are re-sized individually and can be initially built in parallel. For instance, 64 may be a sensible value for 100 million documents.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str | unicode*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name*, *users=None*)

Create a new database.

Parameters

- **name** (*str | unicode*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name*, *edge_definitions=None*, *orphan_collections=None*, *smart=None*, *smart_field=None*, *shard_count=None*)

Create a new graph.

Parameters

- **name** (*str | unicode*) – Graph name.
- **edge_definitions** (*[dict]*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str | unicode]*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | unicode*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name*, *command*, *params=None*, *period=None*, *offset=None*, *task_id=None*)
Create a new server task.

Parameters

- **name** (*str | unicode*) – Name of the server task.
- **command** (*str | unicode*) – Javascript command to execute.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **period** (*int*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int*) – Initial delay before execution in seconds.
- **task_id** (*str | unicode*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises *arango.exceptions.TaskCreateError* – If create fails.

create_user(*username*, *password*, *active=True*, *extra=None*)
Create a new user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – Password.
- **active** (*bool*) – True if user is active, False otherwise.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name*, *view_type*, *properties=None*)
Create a view.

Parameters

- **name** (*str | unicode*) – View name.
- **view_type** (*str | unicode*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases()
Return the names all databases.

Returns Database names.

Return type [str | unicode]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str | unicode

delete_collection(*name*, *ignore_missing=False*, *system=None*)
Delete the collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name*, *ignore_missing=False*)
Delete the database.

Parameters

- **name** (*str | unicode*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *return_old=False*, *sync=None*, *silent=False*)
Delete a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_graph(*name*, *ignore_missing=False*, *drop_collections=None*)
Drop the graph of the given name from the database.

Parameters

- **name** (*str | unicode*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.GraphDeleteError*** – If delete fails.

delete_task(*task_id*, *ignore_missing=False*)
Delete a server task.

Parameters

- **task_id** (*str | unicode*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.TaskDeleteError*** – If delete fails.

delete_user(*username*, *ignore_missing=False*)
Delete a user.

Parameters

- **username** (*str | unicode*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises ***arango.exceptions.UserDeleteError*** – If delete fails.

delete_view(*name*, *ignore_missing=False*)
Delete a view.

Parameters **name** (*str | unicode*) – View name.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type dict

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details()
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document*, *rev=None*, *check_rev=True*)
Return a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- *arango.exceptions.DocumentGetError* – If retrieval fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

echo()
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises *arango.exceptions.ServerEchoError* – If retrieval fails.

endpoints()
Return coordinate endpoints. This method is for clusters only.

Returns List of endpoints.

Return type [str | unicode]

Raises *arango.exceptions.ServerEndpointsError* – If retrieval fails.

engine()
Return the database engine details.

Returns Database engine details.

Return type str | unicode

Raises *arango.exceptions.ServerEngineError* – If retrieval fails.

execute_transaction(*command*, *params=None*, *read=None*, *write=None*, *sync=None*, *timeout=None*, *max_size=None*, *allow_implicit=None*, *intermediate_commit_count=None*, *intermediate_commit_size=None*)

Execute raw Javascript command in transaction.

Parameters

- **command** (*str | unicode*) – Javascript command to execute.
- **read** (*[str | unicode]*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str | unicode]*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int*) – Max transaction size limit in bytes. Applies only to RocksDB storage engine.
- **allow_implicit** (*bool*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.
- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

Returns Return value of **command**.

Return type str | unicode

Raises ***arango.exceptions.TransactionExecuteError*** – If execution fails.

foxx

Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name*)

Return the graph API wrapper.

Parameters **name** (*str | unicode*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs()

List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises ***arango.exceptions.GraphListError*** – If retrieval fails.

has_collection(*name*)
Check if collection exists in the database.

Parameters **name** (*str | unicode*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name*)
Check if a database exists.

Parameters **name** (*str | unicode*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document*, *rev=None*, *check_rev=True*)
Check if a document exists.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name*)
Check if a graph exists in the database.

Parameters **name** (*str | unicode*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username*)
Check if user exists.

Parameters **username** (*str | unicode*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection*, *document*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)
Insert a new document.

Parameters

- **collection** (*str | unicode*) – Collection name.

- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

log_levels()
Return current logging levels.

Returns Current logging levels.

Return type dict

name
Return database name.

Returns Database name.

Return type str | unicode

permission(*username*, *database*, *collection=None*)
Return user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns Permission for given database or collection.

Return type str | unicode

Raise arango.exceptions.PermissionGetError: If retrieval fails.

permissions(*username*)
Return user permissions for all databases and collections.

Parameters **username** (*str | unicode*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raise arango.exceptions.PermissionListError: If retrieval fails.

ping()
Ping the ArangoDB server by sending a test request.

Returns Response code from server.

Return type int

Raises *arango.exceptions.ServerConnectionError* – If ping fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties()
Return database properties.

Returns Database properties.

Return type dict

Raises *arango.exceptions.DatabasePropertiesError* – If retrieval fails.

read_log(*upto=None*, *level=None*, *start=None*, *size=None*, *offset=None*, *search=None*, *sort=None*)
Read the global log from server.

Parameters

- **upto** (*str | unicode | int*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*str | unicode | int*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str | unicode*) – Return only the log entries containing the given text.
- **sort** (*str | unicode*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises *arango.exceptions.ServerReadLogError* – If read fails.

reload_routing()
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises *arango.exceptions.ServerReloadRoutingError* – If reload fails.

rename_view(*name*, *new_name*)
Rename a view.

Parameters **name** (*str | unicode*) – View name.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewRenameError*** – If delete fails.

replace_document(*document*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)

Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username*, *password*, *active=None*, *extra=None*)

Replace a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name*, *properties*)

Replace a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewReplaceError* – If replace fails.

required_db_version()
Return required version of target database.

Returns Required version of target database.

Return type str | unicode

Raises *arango.exceptions.ServerRequiredDBVersionError* – If retrieval fails.

reset_permission(*username*, *database*, *collection=None*)
Reset user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role()
Return server role in cluster.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY" or "UNDEFINED".

Return type str | unicode

Raises *arango.exceptions.ServerRoleError* – If retrieval fails.

run_tests(*tests*)
Run available unittests on the server.

Parameters **tests** (*[str | unicode]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises *arango.exceptions.ServerRunTestsError* – If execution fails.

set_log_levels(***kwargs*)
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown()
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises *arango.exceptions.ServerShutdownError* – If shutdown fails.

statistics(*description=False*)
Return server statistics.

Returns Server statistics.

Return type dict

Raises *arango.exceptions.ServerStatisticsError* – If retrieval fails.

status()
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises *arango.exceptions.ServerStatusError* – If retrieval fails.

task(*task_id*)
Return the details of an active server task.

Parameters **task_id** (*str | unicode*) – Server task ID.

Returns Server task details.

Return type dict

Raises *arango.exceptions.TaskGetError* – If retrieval fails.

tasks()
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises *arango.exceptions.TaskListError* – If retrieval fails.

time()
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises *arango.exceptions.ServerTimeError* – If retrieval fails.

update_document(*document*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username*, *permission*, *database*, *collection=None*)

Update user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.
- **permission** (*str | unicode*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username*, *password=None*, *active=None*, *extra=None*)

Update a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name*, *properties*)

Update a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewUpdateError* – If update fails.

user(*username*)
Return user details.

Parameters **username** (*str | unicode*) – Username.

Returns User details.

Return type dict

Raises *arango.exceptions.UserGetError* – If retrieval fails.

username
Return the username.

Returns Username.

Return type str | unicode

users()
Return all user details.

Returns List of user details.

Return type [dict]

Raises *arango.exceptions.UserListError* – If retrieval fails.

version()
Return ArangoDB server version.

Returns Server version.

Return type str | unicode

Raises *arango.exceptions.ServerVersionError* – If retrieval fails.

view(*name*)
Return view details.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewGetError* – If retrieval fails.

views()
Return list of views.

Returns List of views.

Return type [dict]

Raises *arango.exceptions.ViewListError* – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.24.12 EdgeCollection

class arango.collection.**EdgeCollection**(*connection*, *executor*, *graph*, *name*)
ArangoDB edge collection API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.
- **graph** (*str | unicode*) – Graph name.
- **name** (*str | unicode*) – Edge collection name.

delete(*edge*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *sync=None*)
Delete an edge document.

Parameters

- **edge** (*str | unicode | dict*) – Edge document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns True if edge was deleted successfully, False if edge was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

edges(*vertex*, *direction=None*)
Return the edge documents coming in and/or out of the vertex.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID or body with "_id" field.
- **direction** (*str | unicode*) – The direction of the edges. Allowed values are "in" and "out". If not set, edges in both directions are returned.

Returns List of edges and statistics.

Return type dict

Raises ***arango.exceptions.EdgeListError*** – If retrieval fails.

get(*edge*, *rev=None*, *check_rev=True*)
Return an edge document.

Parameters

- **edge** (*str | unicode | dict*) – Edge document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns Edge document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

graph
Return the graph name.

Returns Graph name.

Return type str | unicode

insert (*edge*, *sync=None*, *silent=False*)
Insert a new edge document.

Parameters

- **edge** (*dict*) – New edge document to insert. It must contain "_from" and "_to" fields. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

link (*from_vertex*, *to_vertex*, *data=None*, *sync=None*, *silent=False*)
Insert a new edge document linking the given vertices.

Parameters

- **from_vertex** (*str | unicode | dict*) – "From" vertex document ID or body with "_id" field.
- **to_vertex** (*str | unicode | dict*) – "To" vertex document ID or body with "_id" field.
- **data** (*dict*) – Any extra data for the new edge document. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated).
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

replace(*edge*, *check_rev=True*, *sync=None*, *silent=False*)
Replace an edge document.

Parameters

- **edge** (*dict*) – New edge document to replace the old one with. It must contain the "_key" or "_id" field. It must also contain the "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- *arango.exceptions.DocumentReplaceError* – If replace fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

update(*edge*, *check_rev=True*, *keep_none=True*, *sync=None*, *silent=False*)
Update an edge document.

Parameters

- **edge** (*dict*) – Partial or full edge document with updated values. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- *arango.exceptions.DocumentUpdateError* – If update fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

3.24.13 Foxx

class arango.foxx.**Foxx**(*connection*, *executor*)
Foxx API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.

commit(*replace=None*)
Commit local service state of the coordinator to the database.

This can be used to resolve service conflicts between coordinators that cannot be fixed automatically due to missing data.

Parameters **replace** (*bool*) – Overwrite any existing service files in database.

Returns True if the state was committed successfully.

Return type bool

Raises *arango.exceptions.FoxxCommitError* – If commit fails.

config(*mount*)
Return service configuration.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Configuration values.

Return type dict

Raises *arango.exceptions.FoxxConfigGetError* – If retrieval fails.

create_service(*mount*, *source*, *config=None*, *dependencies=None*, *development=None*, *setup=None*, *legacy=None*)
Install a new service.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str | unicode*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict*) – Configuration values.
- **dependencies** (*dict*) – Dependency settings.
- **development** (*bool*) – Enable development mode.
- **setup** (*bool*) – Run service setup script.
- **legacy** (*bool*) – Install the service in 2.8 legacy compatibility mode.

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxServiceCreateError* – If install fails.

delete_service(*mount*, *teardown=None*)
Uninstall a service.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

- **teardown** (*bool*) – Run service teardown script.

Returns True if service was deleted successfully.

Return type bool

Raises *arango.exceptions.FoxxServiceDeleteError* – If delete fails.

dependencies(*mount*)

Return service dependencies.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyGetError* – If retrieval fails.

disable_development(*mount*)

Put the service into production mode.

In a cluster with multiple coordinators, the services on all other coordinators are replaced with the version on the calling coordinator.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxDevModeDisableError* – If operation fails.

download(*mount*)

Download service bundle.

When development mode is enabled, a new bundle is created every time. Otherwise, the bundle represents the version of the service installed on the server.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service bundle in raw string form.

Return type str | unicode

Raises *arango.exceptions.FoxxDownloadError* – If download fails.

enable_development(*mount*)

Put the service into development mode.

While the service is running in development mode, it is reloaded from the file system, and its setup script (if any) is re-executed every time the service handles a request.

In a cluster with multiple coordinators, changes to the filesystem on one coordinator is not reflected across other coordinators.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxDevModeEnableError* – If operation fails.

readme(*mount*)

Return the service readme.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service readme.

Return type str | unicode

Raises *arango.exceptions.FoxxReadmeGetError* – If retrieval fails.

replace_config(*mount*, *config*)
Replace service configuration.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **config** (*dict*) – Configuration values. Omitted options are reset to their default values or marked as un-configured.

Returns Replaced configuration values.

Return type dict

Raises *arango.exceptions.FoxxConfigReplaceError* – If replace fails.

replace_dependencies(*mount*, *dependencies*)
Replace service dependencies.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **dependencies** (*dict*) – Dependencies settings. Omitted ones are disabled.

Returns Replaced dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyReplaceError* – If replace fails.

replace_service(*mount*, *source*, *config=None*, *dependencies=None*, *teardown=None*, *setup=None*, *legacy=None*, *force=None*)
Replace a service by removing the old one and installing a new one.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str | unicode*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict*) – Configuration values.
- **dependencies** (*dict*) – Dependency settings.
- **teardown** (*bool*) – Run service teardown script.
- **setup** (*bool*) – Run service setup script.
- **legacy** (*bool*) – Install the service in 2.8 legacy compatibility mode.
- **force** (*bool*) – Force install if no service is found.

Returns Replaced service metadata.

Return type dict

Raises *arango.exceptions.FoxxServiceReplaceError* – If replace fails.

run_script(*mount*, *name*, *arg=None*)
Run a service script.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **name** (*str | unicode*) – Script name.
- **arg** (*str | unicode | bool | int | list | dict*) – Arbitrary value passed into the script as first argument.

Returns Result of the script, if any.

Return type dict

Raises *arango.exceptions.FoxxScriptRunError* – If script fails.

run_tests(*mount*, *reporter=u'default'*, *idiomatic=None*, *output_format=None*)
Run service tests.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **reporter** (*str | unicode*) – Test reporter. Allowed values are "default" (simple list of test cases), "suite" (object of test cases nested in suites), "stream" (raw stream of test results), "xunit" (XUnit or JUnit compatible structure), or "tap" (raw TAP compatible stream).
- **idiomatic** – Use matching format for the reporter, regardless of the value of parameter **output_format**.
- **output_format** (*str | unicode*) – Used to further control format. Allowed values are "x-ldjson", "xml" and "text". When using "stream" reporter, setting this to "x-ldjson" returns newline-delimited JSON stream. When using "tap" reporter, setting this to "text" returns plain text TAP report. When using "xunit" reporter, settings this to "xml" returns an XML instead of JSONML.

Type bool

Returns Reporter output (e.g. raw JSON string, XML, plain text).

Return type str | unicode

Raises *arango.exceptions.FoxxTestRunError* – If test fails.

scripts(*mount*)
List service scripts.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service scripts.

Return type dict

Raises *arango.exceptions.FoxxScriptListError* – If retrieval fails.

service(*mount*)
Return service metadata.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Service metadata.

Return type dict

Raises *arango.exceptions.FoxxServiceGetError* – If retrieval fails.

services(*exclude_system=False*)
List installed services.

Parameters **exclude_system** (*bool*) – If set to True, system services are excluded.

Returns List of installed service.

Return type [dict]

Raises *arango.exceptions.FoxxServiceListError* – If retrieval fails.

swagger(*mount*)
Return the Swagger API description for the given service.

Parameters **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").

Returns Swagger API description.

Return type dict

Raises *arango.exceptions.FoxxSwaggerGetError* – If retrieval fails.

update_config(*mount*, *config*)
Update service configuration.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **config** (*dict*) – Configuration values. Omitted options are ignored.

Returns Updated configuration values.

Return type dict

Raises *arango.exceptions.FoxxConfigUpdateError* – If update fails.

update_dependencies(*mount*, *dependencies*)
Update service dependencies.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **dependencies** (*dict*) – Dependencies settings. Omitted ones are ignored.

Returns Updated dependency settings.

Return type dict

Raises *arango.exceptions.FoxxDependencyUpdateError* – If update fails.

update_service(*mount*, *source=None*, *config=None*, *dependencies=None*, *teardown=None*, *setup=None*, *legacy=None*)
Update (upgrade) a service.

Parameters

- **mount** (*str | unicode*) – Service mount path (e.g "/_admin/aardvark").
- **source** (*str | unicode*) – Fully qualified URL or absolute path on the server file system. Must be accessible by the server, or by all servers if in a cluster.
- **config** (*dict*) – Configuration values.
- **dependencies** (*dict*) – Dependency settings.
- **teardown** (*bool*) – Run service teardown script.
- **setup** (*bool*) – Run service setup script.
- **legacy** (*bool*) – Install the service in 2.8 legacy compatibility mode.

Returns Updated service metadata.

Return type dict

Raises *arango.exceptions.FoxxServiceUpdateError* – If update fails.

3.24.14 Graph

class arango.graph.**Graph**(*connection*, *executor*, *name*)
Graph API wrapper.

Parameters

- **executor** (*arango.executor.Executor*) – API executor.
- **name** (*str | unicode*) – Graph name.

create_edge_definition(*edge_collection*, *from_vertex_collections*, *to_vertex_collections*)
Create a new edge definition.

An edge definition consists of an edge collection, "from" vertex collection(s) and "to" vertex collection(s). Here is an example entry:

```
{
    'edge_collection': 'edge_collection_name',
    'from_vertex_collections': ['from_vertex_collection_name'],
    'to_vertex_collections': ['to_vertex_collection_name']
}
```

Parameters

- **edge_collection** (*str | unicode*) – Edge collection name.
- **from_vertex_collections** (*[str | unicode]*) – Names of "from" vertex collections.
- **to_vertex_collections** (*[str | unicode]*) – Names of "to" vertex collections.

Returns Edge collection API wrapper.

Return type *arango.collection.EdgeCollection*

Raises *arango.exceptions.EdgeDefinitionCreateError* – If create fails.

create_vertex_collection(*name*)
Create a vertex collection in the graph.

Parameters **name** (*str | unicode*) – Vertex collection name.

Returns Vertex collection API wrapper.

Return type *arango.collection.VertexCollection*

Raises *arango.exceptions.VertexCollectionCreateError* – If create fails.

delete_edge(*edge*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *sync=None*)
Delete an edge document.

Parameters

- **edge** (*str | unicode | dict*) – Edge document ID or body with "_id" field.

- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns True if edge was deleted successfully, False if edge was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_edge_definition(*name*, *purge=False*)
Delete an edge definition from the graph.

Parameters

- **name** (*str | unicode*) – Edge collection name.
- **purge** (*bool*) – If set to True, the edge definition is not just removed from the graph but the edge collection is also deleted completely from the database.

Returns True if edge definition was deleted successfully.

Return type bool

Raises ***arango.exceptions.EdgeDefinitionDeleteError*** – If delete fails.

delete_vertex(*vertex*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *sync=None*)
Delete a vertex document.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns True if vertex was deleted successfully, False if vertex was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

delete_vertex_collection(*name*, *purge=False*)
Remove a vertex collection from the graph.

Parameters

- **name** (*str | unicode*) – Vertex collection name.
- **purge** (*bool*) – If set to True, the vertex collection is not just deleted from the graph but also from the database completely.

Returns True if vertex collection was deleted successfully.

Return type bool

Raises ***arango.exceptions.VertexCollectionDeleteError*** – If delete fails.

edge(*edge*, *rev=None*, *check_rev=True*)
Return an edge document.

Parameters

- **edge** (*str | unicode | dict*) – Edge document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns Edge document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

edge_collection(*name*)
Return the edge collection API wrapper.

Parameters **name** (*str | unicode*) – Edge collection name.

Returns Edge collection API wrapper.

Return type *arango.collection.EdgeCollection*

edge_definitions()
Return the edge definitions of the graph.

Returns Edge definitions of the graph.

Return type [dict]

Raises ***arango.exceptions.EdgeDefinitionListError*** – If retrieval fails.

edges(*collection*, *vertex*, *direction=None*)
Return the edge documents coming in and/or out of given vertex.

Parameters

- **collection** (*str | unicode*) – Edge collection name.
- **vertex** (*str | unicode | dict*) – Vertex document ID or body with "_id" field.
- **direction** (*str | unicode*) – The direction of the edges. Allowed values are "in" and "out". If not set, edges in both directions are returned.

Returns List of edges and statistics.

Return type dict

Raises *arango.exceptions.EdgeListError* – If retrieval fails.

has_edge(*edge*, *rev=None*, *check_rev=True*)
Check if the given edge document exists in the graph.

Parameters

- **edge** (*str | unicode | dict*) – Edge document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **edge** if present.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.

Returns True if edge document exists, False otherwise.

Return type bool

Raises

- *arango.exceptions.DocumentInError* – If check fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

has_edge_collection(*name*)
Check if the graph has the given edge collection.

Parameters **name** (*str | unicode*) – Edge collection name.

Returns True if edge collection exists, False otherwise.

Return type bool

has_edge_definition(*name*)
Check if the graph has the given edge definition.

Parameters **name** (*str | unicode*) – Edge collection name.

Returns True if edge definition exists, False otherwise.

Return type bool

has_vertex(*vertex*, *rev=None*, *check_rev=True*)
Check if the given vertex document exists in the graph.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns True if vertex document exists, False otherwise.

Return type bool

Raises

- *arango.exceptions.DocumentGetError* – If check fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

has_vertex_collection(*name*)
Check if the graph has the given vertex collection.

Parameters **name** (*str | unicode*) – Vertex collection name.

Returns True if vertex collection exists, False otherwise.

Return type bool

insert_edge(*collection*, *edge*, *sync=None*, *silent=False*)
Insert a new edge document.

Parameters

- **collection** (*str | unicode*) – Edge collection name.
- **edge** (*dict*) – New edge document to insert. It must contain "_from" and "_to" fields. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

insert_vertex(*collection*, *vertex*, *sync=None*, *silent=False*)
Insert a new vertex document.

Parameters

- **collection** (*str | unicode*) – Vertex collection name.
- **vertex** (*dict*) – New vertex document to insert. If it has "_key" or "_id" field, its value is used as key of the new vertex (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises *arango.exceptions.DocumentInsertError* – If insert fails.

link(*collection*, *from_vertex*, *to_vertex*, *data=None*, *sync=None*, *silent=False*)
Insert a new edge document linking the given vertices.

Parameters

- **collection** (*str | unicode*) – Edge collection name.
- **from_vertex** (*str | unicode | dict*) – "From" vertex document ID or body with "_id" field.
- **to_vertex** (*str | unicode | dict*) – "To" vertex document ID or body with "_id" field.

- **data** (*dict*) – Any extra data for the new edge document. If it has "_key" or "_id" field, its value is used as key of the new edge document (otherwise it is auto-generated).
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

name
Return the graph name.

Returns Graph name.

Return type str | unicode

properties()
Return graph properties.

Returns Graph properties.

Return type dict

Raises ***arango.exceptions.GraphPropertiesError*** – If retrieval fails.

replace_edge(*edge*, *check_rev=True*, *sync=None*, *silent=False*)
Replace an edge document.

Parameters

- **edge** (*dict*) – New edge document to replace the old one with. It must contain the "_id" field. It must also contain the "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_edge_definition(*edge_collection*, *from_vertex_collections*, *to_vertex_collections*)
Replace an edge definition.

Parameters

- **edge_collection** (*str | unicode*) – Edge collection name.
- **from_vertex_collections** (*[str | unicode]*) – Names of "from" vertex collections.

- **to_vertex_collections** (*[str | unicode]*) – Names of "to" vertex collections.

Returns True if edge definition was replaced successfully.

Return type bool

Raises ***arango.exceptions.EdgeDefinitionReplaceError*** – If replace fails.

replace_vertex(*vertex*, *check_rev=True*, *sync=None*, *silent=False*)
Replace a vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to replace the old one with. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

traverse(*start_vertex*, *direction=u'outbound'*, *item_order=u'forward'*, *strategy=None*, *order=None*, *edge_uniqueness=None*, *vertex_uniqueness=None*, *max_iter=None*, *min_depth=None*, *max_depth=None*, *init_func=None*, *sort_func=None*, *filter_func=None*, *visitor_func=None*, *expander_func=None*)
Traverse the graph and return the visited vertices and edges.

Parameters

- **start_vertex** (*str | unicode | dict*) – Start vertex document ID or body with "_id" field.
- **direction** (*str | unicode*) – Traversal direction. Allowed values are "outbound" (default), "inbound" and "any".
- **item_order** (*str | unicode*) – Item iteration order. Allowed values are "forward" (default) and "backward".
- **strategy** (*str | unicode*) – Traversal strategy. Allowed values are "depthfirst" and "breadthfirst".
- **order** (*str | unicode*) – Traversal order. Allowed values are "preorder", "postorder", and "preorder-expander".
- **vertex_uniqueness** (*str | unicode*) – Uniqueness for visited vertices. Allowed values are "global", "path" or "none".
- **edge_uniqueness** (*str | unicode*) – Uniqueness for visited edges. Allowed values are "global", "path" or "none".
- **min_depth** (*int*) – Minimum depth of the nodes to visit.

- **max_depth** (*int*) – Maximum depth of the nodes to visit.
- **max_iter** (*int*) – If set, halt the traversal after the given number of iterations. This parameter can be used to prevent endless loops in cyclic graphs.
- **init_func** (*str | unicode*) – Initialization function in Javascript with signature `(config, result) -> void`. This function is used to initialize values in the result.
- **sort_func** (*str | unicode*) – Sorting function in Javascript with signature `(left, right) -> integer`, which returns `-1` if `left < right`, `+1` if `left > right` and `0` if `left == right`.
- **filter_func** (*str | unicode*) – Filter function in Javascript with signature `(config, vertex, path) -> mixed`, where `mixed` can have one of the following values (or an array with multiple): "exclude" (do not visit the vertex), "prune" (do not follow the edges of the vertex), or "undefined" (visit the vertex and follow its edges).
- **visitor_func** (*str | unicode*) – Visitor function in Javascript with signature `(config, result, vertex, path, connected) -> void`. The return value is ignored, `result` is modified by reference, and `connected` is populated only when parameter **order** is set to "preorder-expander".
- **expander_func** (*str | unicode*) – Expander function in Javascript with signature `(config, vertex, path) -> mixed`. The function must return an array of connections for `vertex`. Each connection is an object with attributes "edge" and "vertex".

Returns Visited edges and vertices.

Return type dict

Raises ***arango.exceptions.GraphTraverseError*** – If traversal fails.

update_edge(*edge*, *check_rev=True*, *keep_none=True*, *sync=None*, *silent=False*)
Update an edge document.

Parameters

- **edge** (*dict*) – Partial or full edge document with updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **edge** (if given) is compared against the revision of target edge document.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_vertex(*vertex*, *check_rev=True*, *keep_none=True*, *sync=None*, *silent=False*)
Update a vertex document.

Parameters

- **vertex** (*dict*) – Partial or full vertex document with updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

vertex(*vertex*, *rev=None*, *check_rev=True*)

Return a vertex document.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns Vertex document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

vertex_collection(*name*)

Return the vertex collection API wrapper.

Parameters **name** (*str | unicode*) – Vertex collection name.

Returns Vertex collection API wrapper.

Return type *arango.collection.VertexCollection*

vertex_collections()

Return vertex collections in the graph that are not orphaned.

Returns Names of vertex collections that are not orphaned.

Return type [str | unicode]

Raises ***arango.exceptions.VertexCollectionListError*** – If retrieval fails.

3.24.15 HTTPClient

class arango.http.**HTTPClient**
Abstract base class for HTTP clients.

send_request(*method*, *url*, *headers=None*, *params=None*, *data=None*, *auth=None*)
Send an HTTP request.

This method must be overridden by the user.

Parameters

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **url** (*str | unicode*) – Request URL.
- **headers** (*dict*) – Request headers.
- **params** (*dict*) – URL (query) parameters.
- **data** (*str | unicode | bool | int | list | dict*) – Request payload.
- **auth** (*tuple*) – Username and password.

Returns HTTP response.

Return type *arango.response.Response*

3.24.16 Pregel

class arango.pregel.**Pregel**(*connection*, *executor*)
Pregel API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.

create_job(*graph*, *algorithm*, *store=True*, *max_gss=None*, *thread_count=None*, *async_mode=None*, *result_field=None*, *algorithm_params=None*)
Start a new Pregel job.

Parameters

- **graph** (*str | unicode*) – Graph name.
- **algorithm** (*str | unicode*) – Algorithm (e.g. "pagerank").
- **store** (*bool*) – If set to True, Pregel engine writes results back to the database. If set to False, results can be queried via AQL.
- **max_gss** (*int*) – Max number of global iterations for the algorithm.
- **thread_count** (*int*) – Number of parallel threads to use per worker. This does not influence the number of threads used to load or store data from the database (it depends on the number of shards).
- **async_mode** (*bool*) – If set to True, algorithms which support async mode run without synchronized global iterations. This might lead to performance increase if there are load imbalances.
- **result_field** (*str | unicode*) – If specified, most algorithms will write their results into this field.

- **algorithm_params** (*dict*) – Additional algorithm parameters.

Returns Pregel job ID.

Return type int

Raises *arango.exceptions.PregelJobCreateError* – If create fails.

delete_job(*job_id*)
Delete a Pregel job.

Parameters **job_id** (*int*) – Pregel job ID.

Returns True if Pregel job was deleted successfully.

Return type bool

Raises *arango.exceptions.PregelJobDeleteError* – If delete fails.

job(*job_id*)
Return the details of a Pregel job.

Parameters **job_id** (*int*) – Pregel job ID.

Returns Details of the Pregel job.

Return type dict

Raises *arango.exceptions.PregelJobGetError* – If retrieval fails.

3.24.17 Request

class arango.request.**Request**(*method*, *endpoint*, *headers=None*, *params=None*, *data=None*, *command=None*, *read=None*, *write=None*)
HTTP request.

Parameters

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **endpoint** (*str | unicode*) – API endpoint.
- **headers** (*dict*) – Request headers.
- **params** (*dict*) – URL parameters.
- **data** (*str | unicode | bool | int | list | dict*) – Request payload.
- **command** (*str | unicode*) – ArangoSh command.
- **read** (*str | unicode | [str | unicode]*) – Names of collections read during transaction.
- **write** (*str | unicode | [str | unicode]*) – Names of collections written to during transaction.

Variables

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **endpoint** (*str | unicode*) – API endpoint.
- **headers** (*dict*) – Request headers.
- **params** (*dict*) – URL (query) parameters.
- **data** (*str | unicode | bool | int | list | dict*) – Request payload.

- **command** (*str | unicode | None*) – ArangoSh command.
- **read** (*str | unicode | [str | unicode] | None*) – Names of collections read during transaction.
- **write** (*str | unicode | [str | unicode] | None*) – Names of collections written to during transaction.

3.24.18 Response

class arango.response.**Response**(*method*, *url*, *headers*, *status_code*, *status_text*, *raw_body*)

HTTP response.

Parameters

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **url** (*str | unicode*) – API URL.
- **headers** (*requests.structures.CaseInsensitiveDict | dict*) – Response headers.
- **status_code** (*int*) – Response status code.
- **status_text** (*str | unicode*) – Response status text.
- **raw_body** (*str | unicode*) – Raw response body.

Variables

- **method** (*str | unicode*) – HTTP method in lowercase (e.g. "post").
- **url** (*str | unicode*) – API URL.
- **headers** (*requests.structures.CaseInsensitiveDict | dict*) – Response headers.
- **status_code** (*int*) – Response status code.
- **status_text** (*str | unicode*) – Response status text.
- **body** (*str | unicode | bool | int | list | dict*) – JSON-deserialized response body.
- **raw_body** (*str | unicode*) – Raw response body.
- **error_code** (*int*) – Error code from ArangoDB server.
- **error_message** (*str | unicode*) – Error message from ArangoDB server.
- **is_success** (*bool*) – True if response status code was 2XX.

3.24.19 TransactionDatabase

class arango.database.**TransactionDatabase**(*connection*, *return_result*, *read*, *write*, *timeout*, *sync*)

Database API wrapper tailored specifically for transactions.

See *arango.database.StandardDatabase.begin_transaction()*.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.

- **return_result** (*bool*) – If set to True, API executions return instances of *arango.job.TransactionJob* that are populated with results on commit. If set to False, API executions return None and no results are tracked client-side.
- **read** (*[str | unicode]*) – Names of collections read during transaction.
- **write** (*[str | unicode]*) – Names of collections written to during transaction.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, the ArangoDB server waits indefinitely. If not set, system default value is used.
- **sync** (*bool*) – Block until operation is synchronized to disk.

aql
Return AQL (ArangoDB Query Language) API wrapper.

Returns AQL API wrapper.

Return type *arango.aql.AQL*

async_jobs(*status*, *count=None*)
Return IDs of async jobs with given status.

Parameters

- **status** (*str | unicode*) – Job status (e.g. "pending", "done").
- **count** (*int*) – Max number of job IDs to return.

Returns List of job IDs.

Return type [str | unicode]

Raises ***arango.exceptions.AsyncJobListError*** – If retrieval fails.

clear_async_jobs(*threshold=None*)
Clear async job results from the server.

Async jobs that are still queued or running are not stopped.

Parameters **threshold** (*int*) – If specified, only the job results created prior to the threshold (a unix timestamp) are deleted. Otherwise, all job results are deleted.

Returns True if job results were cleared successfully.

Return type bool

Raises ***arango.exceptions.AsyncJobClearError*** – If operation fails.

collection(*name*)
Return the standard collection API wrapper.

Parameters **name** (*str | unicode*) – Collection name.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

collections()
Return the collections in the database.

Returns Collections in the database and their details.

Return type [dict]

Raises ***arango.exceptions.CollectionListError*** – If retrieval fails.

commit()
Execute the queued requests in a single transaction API request.

If **return_result** parameter was set to True during initialization, *arango.job.TransactionJob* instances are populated with results.

Returns Transaction jobs, or None if **return_result** parameter was set to False during initialization.

Return type [*arango.job.TransactionJob*] | None

Raises

- ***arango.exceptions.TransactionStateError*** – If the transaction was already committed.
- ***arango.exceptions.TransactionExecuteError*** – If commit fails.

context
Return the API execution context.

Returns API execution context. Possible values are "default", "async", "batch" and "transaction".

Return type str | unicode

create_collection(*name*, *sync=False*, *compact=True*, *system=False*, *journal_size=None*, *edge=False*, *volatile=False*, *user_keys=True*, *key_increment=None*, *key_offset=None*, *key_generator=u'traditional'*, *shard_fields=None*, *shard_count=None*, *index_bucket_count=None*, *replication_factor=None*, *shard_like=None*, *sync_replication=None*, *enforce_replication_factor=None*)
Create a new collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **sync** (*bool*) – If set to True, document operations via the collection will block until synchronized to disk by default.
- **compact** (*bool*) – If set to True, the collection is compacted. Applies only to MMFiles storage engine.
- **system** (*bool*) – If set to True, a system collection is created. The collection name must have leading underscore "_" character.
- **journal_size** (*int*) – Max size of the journal in bytes.
- **edge** (*bool*) – If set to True, an edge collection is created.
- **volatile** (*bool*) – If set to True, collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data.
- **key_generator** (*str | unicode*) – Used for generating document keys. Allowed values are "traditional" or "autoincrement".
- **user_keys** (*bool*) – If set to True, users are allowed to supply document keys. If set to False, the key generator is solely responsible for supplying the key values.
- **key_increment** (*int*) – Key increment value. Applies only when value of **key_generator** is set to "autoincrement".
- **key_offset** (*int*) – Key offset value. Applies only when value of **key_generator** is set to "autoincrement".

- **shard_fields** (*[str | unicode]*) – Field(s) used to determine the target shard.
- **shard_count** (*int*) – Number of shards to create.
- **index_bucket_count** (*int*) – Number of buckets into which indexes using hash tables are split. The default is 16, and this number has to be a power of 2 and less than or equal to 1024. For large collections, one should increase this to avoid long pauses when the hash table has to be initially built or re-sized, since buckets are re-sized individually and can be initially built in parallel. For instance, 64 may be a sensible value for 100 million documents.
- **replication_factor** (*int*) – Number of copies of each shard on different servers in a cluster. Allowed values are 1 (only one copy is kept and no synchronous replication), and n (n-1 replicas are kept and any two copies are replicated across servers synchronously, meaning every write to the master is copied to all slaves before operation is reported successful).
- **shard_like** (*str | unicode*) – Name of prototype collection whose sharding specifics are imitated. Prototype collections cannot be dropped before imitating collections. Applies to enterprise version of ArangoDB only.
- **sync_replication** (*bool*) – If set to True, server reports success only when collection is created in all replicas. You can set this to False for faster server response, and if full replication is not a concern.
- **enforce_replication_factor** (*bool*) – Check if there are enough replicas available at creation time, or halt the operation.

Returns Standard collection API wrapper.

Return type *arango.collection.StandardCollection*

Raises ***arango.exceptions.CollectionCreateError*** – If create fails.

create_database(*name*, *users=None*)

Create a new database.

Parameters

- **name** (*str | unicode*) – Database name.
- **users** (*[dict]*) – List of users with access to the new database, where each user is a dictionary with fields "username", "password", "active" and "extra" (see below for example). If not set, only the admin and current user are granted access.

Returns True if database was created successfully.

Return type bool

Raises ***arango.exceptions.DatabaseCreateError*** – If create fails.

Here is an example entry for parameter **users**:

```
{
    'username': 'john',
    'password': 'password',
    'active': True,
    'extra': {'Department': 'IT'}
}
```

create_graph(*name*, *edge_definitions=None*, *orphan_collections=None*, *smart=None*, *smart_field=None*, *shard_count=None*)

Create a new graph.

Parameters

- **name** (*str | unicode*) – Graph name.
- **edge_definitions** (*[dict]*) – List of edge definitions, where each edge definition entry is a dictionary with fields "edge_collection", "from_vertex_collections" and "to_vertex_collections" (see below for example).
- **orphan_collections** (*[str | unicode]*) – Names of additional vertex collections that are not in edge definitions.
- **smart** (*bool*) – If set to True, sharding is enabled (see parameter **smart_field** below). Applies only to enterprise version of ArangoDB.
- **smart_field** (*str | unicode*) – Document field used to shard the vertices of the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. Applies only to enterprise version of ArangoDB.
- **shard_count** (*int*) – Number of shards used for every collection in the graph. To use this, parameter **smart** must be set to True and every vertex in the graph must have the smart field. This number cannot be modified later once set. Applies only to enterprise version of ArangoDB.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

Raises ***arango.exceptions.GraphCreateError*** – If create fails.

Here is an example entry for parameter **edge_definitions**:

```
{
    'edge_collection': 'teach',
    'from_vertex_collections': ['teachers'],
    'to_vertex_collections': ['lectures']
}
```

create_task(*name*, *command*, *params=None*, *period=None*, *offset=None*, *task_id=None*)

Create a new server task.

Parameters

- **name** (*str | unicode*) – Name of the server task.
- **command** (*str | unicode*) – Javascript command to execute.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **period** (*int*) – Number of seconds to wait between executions. If set to 0, the new task will be "timed", meaning it will execute only once and be deleted afterwards.
- **offset** (*int*) – Initial delay before execution in seconds.
- **task_id** (*str | unicode*) – Pre-defined ID for the new server task.

Returns Details of the new task.

Return type dict

Raises ***arango.exceptions.TaskCreateError*** – If create fails.

create_user(*username*, *password*, *active=True*, *extra=None*)

Create a new user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – Password.
- **active** (*bool*) – True if user is active, False otherwise.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises *arango.exceptions.UserCreateError* – If create fails.

create_view(*name*, *view_type*, *properties=None*)
Create a view.

Parameters

- **name** (*str | unicode*) – View name.
- **view_type** (*str | unicode*) – View type (e.g. "arangosearch").
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises *arango.exceptions.ViewCreateError* – If create fails.

databases()
Return the names all databases.

Returns Database names.

Return type [str | unicode]

Raises *arango.exceptions.DatabaseListError* – If retrieval fails.

db_name
Return the name of the current database.

Returns Database name.

Return type str | unicode

delete_collection(*name*, *ignore_missing=False*, *system=None*)
Delete the collection.

Parameters

- **name** (*str | unicode*) – Collection name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing collection.
- **system** (*bool*) – Whether the collection is a system collection.

Returns True if collection was deleted successfully, False if collection was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.CollectionDeleteError* – If delete fails.

delete_database(*name*, *ignore_missing=False*)
Delete the database.

Parameters

- **name** (*str | unicode*) – Database name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing database.

Returns True if database was deleted successfully, False if database was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.DatabaseDeleteError* – If delete fails.

delete_document(*document*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *return_old=False*, *sync=None*, *silent=False*)

Delete a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID, key or body. Document body must contain the "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision), or True if parameter **silent** was set to True, or False if document was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool | dict

Raises

- *arango.exceptions.DocumentDeleteError* – If delete fails.
- *arango.exceptions.DocumentRevisionError* – If revisions mismatch.

delete_graph(*name*, *ignore_missing=False*, *drop_collections=None*)

Drop the graph of the given name from the database.

Parameters

- **name** (*str | unicode*) – Graph name.
- **ignore_missing** (*bool*) – Do not raise an exception on missing graph.
- **drop_collections** (*bool*) – Drop the collections of the graph also. This is only if they are not in use by other graphs.

Returns True if graph was deleted successfully, False if graph was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.GraphDeleteError* – If delete fails.

delete_task(*task_id*, *ignore_missing=False*)
Delete a server task.

Parameters

- **task_id** (*str | unicode*) – Server task ID.
- **ignore_missing** (*bool*) – Do not raise an exception on missing task.

Returns True if task was successfully deleted, False if task was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.TaskDeleteError* – If delete fails.

delete_user(*username*, *ignore_missing=False*)
Delete a user.

Parameters

- **username** (*str | unicode*) – Username.
- **ignore_missing** (*bool*) – Do not raise an exception on missing user.

Returns True if user was deleted successfully, False if user was not found and **ignore_missing** was set to True.

Return type bool

Raises *arango.exceptions.UserDeleteError* – If delete fails.

delete_view(*name*, *ignore_missing=False*)
Delete a view.

Parameters **name** (*str | unicode*) – View name.

Returns True if view was deleted successfully, False if view was not found and **ignore_missing** was set to True.

Return type dict

Raises *arango.exceptions.ViewDeleteError* – If delete fails.

details()
Return ArangoDB server details.

Returns Server details.

Return type dict

Raises *arango.exceptions.ServerDetailsError* – If retrieval fails.

document(*document*, *rev=None*, *check_rev=True*)
Return a document.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns Document, or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

echo()
Return details of the last request (e.g. headers, payload).

Returns Details of the last request.

Return type dict

Raises ***arango.exceptions.ServerEchoError*** – If retrieval fails.

endpoints()
Return coordinate endpoints. This method is for clusters only.

Returns List of endpoints.

Return type [str | unicode]

Raises ***arango.exceptions.ServerEndpointsError*** – If retrieval fails.

engine()
Return the database engine details.

Returns Database engine details.

Return type str | unicode

Raises ***arango.exceptions.ServerEngineError*** – If retrieval fails.

execute_transaction(*command*, *params=None*, *read=None*, *write=None*, *sync=None*, *timeout=None*, *max_size=None*, *allow_implicit=None*, *intermediate_commit_count=None*, *intermediate_commit_size=None*)
Execute raw Javascript command in transaction.

Parameters

- **command** (*str | unicode*) – Javascript command to execute.
- **read** (*[str | unicode]*) – Names of collections read during transaction. If parameter **allow_implicit** is set to True, any undeclared read collections are loaded lazily.
- **write** (*[str | unicode]*) – Names of collections written to during transaction. Transaction fails on undeclared write collections.
- **params** (*dict*) – Optional parameters passed into the Javascript command.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **timeout** (*int*) – Timeout for waiting on collection locks. If set to 0, ArangoDB server waits indefinitely. If not set, system default value is used.
- **max_size** (*int*) – Max transaction size limit in bytes. Applies only to RocksDB storage engine.
- **allow_implicit** (*bool*) – If set to True, undeclared read collections are loaded lazily. If set to False, transaction fails on any undeclared collections.
- **intermediate_commit_count** (*int*) – Max number of operations after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

- **intermediate_commit_size** (*int*) – Max size of operations in bytes after which an intermediate commit is performed automatically. Applies only to RocksDB storage engine.

Returns Return value of **command**.

Return type str | unicode

Raises *arango.exceptions.TransactionExecuteError* – If execution fails.

foxx
Return Foxx API wrapper.

Returns Foxx API wrapper.

Return type *arango.foxx.Foxx*

graph(*name*)
Return the graph API wrapper.

Parameters **name** (*str | unicode*) – Graph name.

Returns Graph API wrapper.

Return type *arango.graph.Graph*

graphs()
List all graphs in the database.

Returns Graphs in the database.

Return type [dict]

Raises *arango.exceptions.GraphListError* – If retrieval fails.

has_collection(*name*)
Check if collection exists in the database.

Parameters **name** (*str | unicode*) – Collection name.

Returns True if collection exists, False otherwise.

Return type bool

has_database(*name*)
Check if a database exists.

Parameters **name** (*str | unicode*) – Database name.

Returns True if database exists, False otherwise.

Return type bool

has_document(*document*, *rev=None*, *check_rev=True*)
Check if a document exists.

Parameters

- **document** (*str | unicode | dict*) – Document ID or body with "_id" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides value of "_rev" field in **document** if present.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.

Returns True if document exists, False otherwise.

Return type bool

Raises

- ***arango.exceptions.DocumentInError*** – If check fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

has_graph(*name*)
Check if a graph exists in the database.

Parameters **name** (*str | unicode*) – Graph name.

Returns True if graph exists, False otherwise.

Return type bool

has_user(*username*)
Check if user exists.

Parameters **username** (*str | unicode*) – Username.

Returns True if user exists, False otherwise.

Return type bool

insert_document(*collection*, *document*, *return_new=False*, *sync=None*, *silent=False*, *overwrite=False*, *return_old=False*)
Insert a new document.

Parameters

- **collection** (*str | unicode*) – Collection name.
- **document** (*dict*) – Document to insert. If it contains the "_key" or "_id" field, the value is used as the key of the new document (otherwise it is auto-generated). Any "_rev" field is ignored.
- **return_new** (*bool*) – Include body of the new document in the returned metadata. Ignored if parameter **silent** is set to True.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.
- **overwrite** (*bool*) – If set to True, operation does not fail on duplicate key and the existing document is replaced.
- **return_old** (*bool*) – Include body of the old document if replaced. Applies only when value of **overwrite** is set to True.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

log_levels()
Return current logging levels.

Returns Current logging levels.

Return type dict

name
Return database name.

Returns Database name.

Return type str | unicode

permission(*username*, *database*, *collection=None*)
Return user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns Permission for given database or collection.

Return type str | unicode

Raise arango.exceptions.PermissionGetError: If retrieval fails.

permissions(*username*)
Return user permissions for all databases and collections.

Parameters **username** (*str | unicode*) – Username.

Returns User permissions for all databases and collections.

Return type dict

Raise arango.exceptions.PermissionListError: If retrieval fails.

ping()
Ping the ArangoDB server by sending a test request.

Returns Response code from server.

Return type int

Raises ***arango.exceptions.ServerConnectionError*** – If ping fails.

pregel
Return Pregel API wrapper.

Returns Pregel API wrapper.

Return type *arango.pregel.Pregel*

properties()
Return database properties.

Returns Database properties.

Return type dict

Raises ***arango.exceptions.DatabasePropertiesError*** – If retrieval fails.

queued_jobs()
Return the queued transaction jobs.

Returns Queued transaction jobs, or None if **return_result** was set to False during initialization.

Return type [*arango.job.TransactionJob*] | None

read_log(*upto=None*, *level=None*, *start=None*, *size=None*, *offset=None*, *search=None*, *sort=None*)
Read the global log from server.

Parameters

- **upto** (*str | unicode | int*) – Return the log entries up to the given level (mutually exclusive with parameter **level**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **level** (*str | unicode | int*) – Return the log entries of only the given level (mutually exclusive with **upto**). Allowed values are "fatal", "error", "warning", "info" (default) and "debug".
- **start** (*int*) – Return the log entries whose ID is greater or equal to the given value.
- **size** (*int*) – Restrict the size of the result to the given value. This can be used for pagination.
- **offset** (*int*) – Number of entries to skip (e.g. for pagination).
- **search** (*str | unicode*) – Return only the log entries containing the given text.
- **sort** (*str | unicode*) – Sort the log entries according to the given fashion, which can be "sort" or "desc".

Returns Server log entries.

Return type dict

Raises ***arango.exceptions.ServerReadLogError*** – If read fails.

reload_routing()
Reload the routing information.

Returns True if routing was reloaded successfully.

Return type bool

Raises ***arango.exceptions.ServerReloadRoutingError*** – If reload fails.

rename_view(*name*, *new_name*)
Rename a view.

Parameters **name** (*str | unicode*) – View name.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewRenameError*** – If delete fails.

replace_document(*document*, *check_rev=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Replace a document.

Parameters

- **document** (*dict*) – New document to replace the old one with. It must contain the "_id" field. Edge document must also have "_from" and "_to" fields.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

replace_user(*username*, *password*, *active=None*, *extra=None*)
Replace a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserReplaceError*** – If replace fails.

replace_view(*name*, *properties*)
Replace a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewReplaceError*** – If replace fails.

required_db_version()
Return required version of target database.

Returns Required version of target database.

Return type str | unicode

Raises ***arango.exceptions.ServerRequiredDBVersionError*** – If retrieval fails.

reset_permission(*username*, *database*, *collection=None*)
Reset user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.

Returns True if permission was reset successfully.

Return type bool

Raises **arango.exceptions.PermissionRestError** – If reset fails.

role()
Return server role in cluster.

Returns Server role. Possible values are "SINGLE" (server which is not in a cluster), "COORDINATOR" (cluster coordinator), "PRIMARY", "SECONDARY" or "UNDEFINED".

Return type str | unicode

Raises *arango.exceptions.ServerRoleError* – If retrieval fails.

run_tests(*tests*)
Run available unittests on the server.

Parameters **tests** (*[str | unicode]*) – List of files containing the test suites.

Returns Test results.

Return type dict

Raises *arango.exceptions.ServerRunTestsError* – If execution fails.

set_log_levels(**kwargs*)
Set the logging levels.

This method takes arbitrary keyword arguments where the keys are the logger names and the values are the logging levels. For example:

```
arango.set_log_levels(
    agency='DEBUG',
    collector='INFO',
    threads='WARNING'
)
```

Keys that are not valid logger names are ignored.

Returns New logging levels.

Return type dict

shutdown()
Initiate server shutdown sequence.

Returns True if the server was shutdown successfully.

Return type bool

Raises *arango.exceptions.ServerShutdownError* – If shutdown fails.

statistics(*description=False*)
Return server statistics.

Returns Server statistics.

Return type dict

Raises *arango.exceptions.ServerStatisticsError* – If retrieval fails.

status()
Return ArangoDB server status.

Returns Server status.

Return type dict

Raises *arango.exceptions.ServerStatusError* – If retrieval fails.

task(*task_id*)
Return the details of an active server task.

Parameters **task_id** (*str | unicode*) – Server task ID.

Returns Server task details.

Return type dict

Raises ***arango.exceptions.TaskGetError*** – If retrieval fails.

tasks()
Return all currently active server tasks.

Returns Currently active server tasks.

Return type [dict]

Raises ***arango.exceptions.TaskListError*** – If retrieval fails.

time()
Return server system time.

Returns Server system time.

Return type datetime.datetime

Raises ***arango.exceptions.ServerTimeError*** – If retrieval fails.

update_document(*document*, *check_rev=True*, *merge=True*, *keep_none=True*, *return_new=False*, *return_old=False*, *sync=None*, *silent=False*)
Update a document.

Parameters

- **document** (*dict*) – Partial or full document with the updated values. It must contain the "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **document** (if given) is compared against the revision of target document.
- **merge** (*bool*) – If set to True, sub-dictionaries are merged instead of the new one overwriting the old one.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. Otherwise, they are removed completely.
- **return_new** (*bool*) – Include body of the new document in the result.
- **return_old** (*bool*) – Include body of the old document in the result.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update_permission(*username*, *permission*, *database*, *collection=None*)
Update user permission for a specific database or collection.

Parameters

- **username** (*str | unicode*) – Username.
- **database** (*str | unicode*) – Database name.
- **collection** (*str | unicode*) – Collection name.
- **permission** (*str | unicode*) – Allowed values are "rw" (read and write), "ro" (read only) or "none" (no access).

Returns True if access was granted successfully.

Return type bool

Raises ***arango.exceptions.PermissionUpdateError*** – If update fails.

update_user(*username*, *password=None*, *active=None*, *extra=None*)
Update a user.

Parameters

- **username** (*str | unicode*) – Username.
- **password** (*str | unicode*) – New password.
- **active** (*bool*) – Whether the user is active.
- **extra** (*dict*) – Additional data for the user.

Returns New user details.

Return type dict

Raises ***arango.exceptions.UserUpdateError*** – If update fails.

update_view(*name*, *properties*)
Update a view.

Parameters

- **name** (*str | unicode*) – View name.
- **properties** (*dict*) – View properties.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewUpdateError*** – If update fails.

user(*username*)
Return user details.

Parameters **username** (*str | unicode*) – Username.

Returns User details.

Return type dict

Raises ***arango.exceptions.UserGetError*** – If retrieval fails.

username
Return the username.

Returns Username.

Return type str | unicode

users()
Return all user details.

Returns List of user details.

Return type [dict]

Raises ***arango.exceptions.UserListError*** – If retrieval fails.

version()
Return ArangoDB server version.

Returns Server version.

Return type str | unicode

Raises ***arango.exceptions.ServerVersionError*** – If retrieval fails.

view(*name*)
Return view details.

Returns View details.

Return type dict

Raises ***arango.exceptions.ViewGetError*** – If retrieval fails.

views()
Return list of views.

Returns List of views.

Return type [dict]

Raises ***arango.exceptions.ViewListError*** – If retrieval fails.

wal
Return WAL (Write-Ahead Log) API wrapper.

Returns WAL API wrapper.

Return type *arango.wal.WAL*

3.24.20 TransactionJob

class arango.job.**TransactionJob**(*response_handler*)
Transaction API execution job.

Parameters **response_handler** (*callable*) – HTTP response handler.

id
Return the transaction job ID.

Returns Transaction job ID.

Return type str | unicode

result()
Return the transaction job result.

Returns Transaction job result.

Return type str | unicode | bool | int | list | dict

Raises

- ***arango.exceptions.ArangoError*** – If the job raised an exception.
- ***arango.exceptions.TransactionJobResultError*** – If job result is not available (i.e. transaction is not committed yet or failed).

status()
Return the transaction job status.

Returns Transaction job status. Possible values are "pending" (job is waiting for transaction to be committed, or transaction failed and job is orphaned), or "done" (transaction was committed and job is updated with the result).

Return type str | unicode

3.24.21 VertexCollection

class arango.collection.**VertexCollection**(*connection*, *executor*, *graph*, *name*)
Vertex collection API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.
- **graph** (*str | unicode*) – Graph name.
- **name** (*str | unicode*) – Vertex collection name.

delete(*vertex*, *rev=None*, *check_rev=True*, *ignore_missing=False*, *sync=None*)
Delete a vertex document.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **ignore_missing** (*bool*) – Do not raise an exception on missing document. This parameter has no effect in transactions where an exception is always raised on failures.
- **sync** (*bool*) – Block until operation is synchronized to disk.

Returns True if vertex was deleted successfully, False if vertex was not found and **ignore_missing** was set to True (does not apply in transactions).

Return type bool

Raises

- ***arango.exceptions.DocumentDeleteError*** – If delete fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

get(*vertex*, *rev=None*, *check_rev=True*)
Return a vertex document.

Parameters

- **vertex** (*str | unicode | dict*) – Vertex document ID, key or body. Document body must contain the "_id" or "_key" field.
- **rev** (*str | unicode*) – Expected document revision. Overrides the value of "_rev" field in **vertex** if present.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.

Returns Vertex document or None if not found.

Return type dict | None

Raises

- ***arango.exceptions.DocumentGetError*** – If retrieval fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

graph
Return the graph name.

Returns Graph name.

Return type str | unicode

insert (*vertex*, *sync=None*, *silent=False*)
Insert a new vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to insert. If it has "_key" or "_id" field, its value is used as key of the new vertex (otherwise it is auto-generated). Any "_rev" field is ignored.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises ***arango.exceptions.DocumentInsertError*** – If insert fails.

replace (*vertex*, *check_rev=True*, *sync=None*, *silent=False*)
Replace a vertex document.

Parameters

- **vertex** (*dict*) – New vertex document to replace the old one with. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentReplaceError*** – If replace fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

update(*vertex*, *check_rev=True*, *keep_none=True*, *sync=None*, *silent=False*, *return_old=False*, *return_new=False*)
Update a vertex document.

Parameters

- **vertex** (*dict*) – Partial or full vertex document with updated values. It must contain the "_key" or "_id" field.
- **check_rev** (*bool*) – If set to True, revision of **vertex** (if given) is compared against the revision of target vertex document.
- **keep_none** (*bool*) – If set to True, fields with value None are retained in the document. If set to False, they are removed completely.
- **sync** (*bool*) – Block until operation is synchronized to disk.
- **silent** (*bool*) – If set to True, no document metadata is returned. This can be used to save resources.

Returns Document metadata (e.g. document key, revision) or True if parameter **silent** was set to True.

Return type bool | dict

Raises

- ***arango.exceptions.DocumentUpdateError*** – If update fails.
- ***arango.exceptions.DocumentRevisionError*** – If revisions mismatch.

3.24.22 WAL

class arango.wal.**WAL**(*connection*, *executor*)
WAL (Write-Ahead Log) API wrapper.

Parameters

- **connection** (*arango.connection.Connection*) – HTTP connection.
- **executor** (*arango.executor.Executor*) – API executor.

configure(*oversized_ops=None*, *log_size=None*, *historic_logs=None*, *reserve_logs=None*, *throttle_wait=None*, *throttle_limit=None*)
Configure WAL properties.

Parameters

- **oversized_ops** (*bool*) – If set to True, operations bigger than a single log file are allowed to be executed and stored.
- **log_size** (*int*) – Size of each write-ahead log file in bytes.
- **historic_logs** (*int*) – Max number of historic log files to keep.
- **reserve_logs** (*int*) – Max number of reserve log files to allocate.
- **throttle_wait** (*int*) – Wait time before aborting when write-throttled in milliseconds.

- **throttle_limit** (*int*) – Number of pending garbage collector operations that, when reached, activates write-throttling. Value of 0 means no throttling is triggered.

Returns New WAL properties.

Return type dict

Raises *arango.exceptions.WALConfigureError* – If operation fails.

flush(*sync=True*, *garbage_collect=True*)
Synchronize WAL to disk.

Parameters

- **sync** (*bool*) – Block until the synchronization is complete.
- **garbage_collect** (*bool*) – Block until flushed data is garbage collected.

Returns True if WAL was flushed successfully.

Return type bool

Raises *arango.exceptions.WALFlushError* – If flush operation fails.

properties()
Return WAL properties.

Returns WAL properties.

Return type dict

Raises *arango.exceptions.WALPropertiesError* – If retrieval fails.

transactions()
Return details on currently running WAL transactions.

Fields in the returned details are as follows:

```
"last_collected"    : ID of the last collected log file (at the
                      start of each running transaction) or None
                      if no transactions are running.

"last_sealed"       : ID of the last sealed log file (at the start
                      of each running transaction) or None if no
                      transactions are running.

"count"             : Number of currently running transactions.
```

Returns Details on currently running WAL transactions.

Return type dict

Raises *arango.exceptions.WALTransactionListError* – If retrieval fails.

Python Module Index

a

Index

D

E

F

G

H

I

S

T

U

V

W