📖 API Cheatsheet

ⓘ **NOTE**

This is a quick cheatsheet of the API. For full API docs, refer to the JS and Python docs in the sidebar.

Select a language

Python JavaScript

Initialize client - Python

In-memory chroma

```python
import chromadb
client = chromadb.Client()
```

In-memory chroma with saving/loading to disk

In this mode, Chroma will persist data between sessions. On load - it will load up the data in the directory you specify. And as you add data - it will save to that directory.

```python
import chromadb
client = chromadb.PersistentClient(path="/path/to/data")
```

Run chroma just as a client to talk to a backend service

For many use cases, an in-memory database will not cut it. Run `docker-compose up -d --build` to run a persistent backend in Docker. Simply update your API initialization and then use the API the same way as before.

```python
import chromadb
chroma_client = chromadb.HttpClient(host="localhost", port=8000)
```

Methods on Client

Methods related to Collections

COLLECTION NAMING

Collections are similar to AWS s3 buckets in their naming requirements because they are used in URLs in the REST API. Here's the **full list**.

```
# list all collections
client.list_collections()

# make a new collection
collection = client.create_collection("testname")

# get an existing collection
collection = client.get_collection("testname")

# get a collection or create if it doesn't exist already
collection = client.get_or_create_collection("testname")

# delete a collection
client.delete_collection("testname")
```

Utility methods

```
# resets entire database - this *cant* be undone!
client.reset()

# returns timestamp to check if service is up
client.heartbeat()
```

Methods on Collection

```
# change the name or metadata on a collection
collection.modify(name="testname2")

# get the number of items in a collection
collection.count()

# add new items to a collection
# either one at a time
collection.add(
    embeddings=[1.5, 2.9, 3.4],
    metadatas={"uri": "img9.png", "style": "style1"},
    documents="doc1000101",
    ids="uri9",
)
# or many, up to 100k+!
collection.add(
    embeddings=[[1.5, 2.9, 3.4], [9.8, 2.3, 2.9]],
    metadatas=[{"style": "style1"}, {"style": "style2"}],
    ids=["uri9", "uri10"],
)
collection.add(
    documents=["doc1000101", "doc288822"],
    metadatas=[{"style": "style1"}, {"style": "style2"}],
    ids=["uri9", "uri10"],
)

# update items in a collection
collection.update()

# upsert items. new items will be added, existing items will be updated.
collection.upsert(
    ids=["id1", "id2", "id3", ...],
    embeddings=[[1.1, 2.3, 3.2], [4.5, 6.9, 4.4], [1.1, 2.3, 3.2], ...],
    metadatas=[{"chapter": "3", "verse": "16"}, {"chapter": "3", "verse": "5"},
{"chapter": "29", "verse": "11"}, ...],
    documents=["doc1", "doc2", "doc3", ...],
)

# get items from a collection
collection.get()

# convenience, get first 5 items from a collection
collection.peek()

# do nearest neighbor search to find similar embeddings or documents, supports
filtering
collection.query(
    query_embeddings=[[1.1, 2.3, 3.2], [5.1, 4.3, 2.2]],
    n_results=2,
    where={"style": "style2"}
)

# delete items
collection.delete()
```

Edit this page