Google Cloud Bigtable Documentation

Release 0.0.1

Google Cloud Platform

August 18, 2015

Contents

1 Long-lived Defaults **3**

2 Authorization **5**

3 Project ID **7**

4 Admin API Access **9**

5 Read-Only Mode **11**

6 Next Step **13**

7 Client **15**

8 Cluster Admin API **21**
- 8.1 List Clusters . . . 21
- 8.2 List Zones . . . 21
- 8.3 Cluster Factory . . . 21
- 8.4 Create a new Cluster . . . 21
- 8.5 Check on Current Operation . . . 22
- 8.6 Get metadata for an existing Cluster . . . 22
- 8.7 Update an existing Cluster . . . 22
- 8.8 Delete an existing Cluster . . . 22
- 8.9 Undelete a deleted Cluster . . . 23
- 8.10 Next Step . . . 23

9 Cluster **25**

10 Table Admin API **29**
- 10.1 List Tables . . . 29
- 10.2 Table Factory . . . 29
- 10.3 Create a new Table . . . 29
- 10.4 Delete an existing Table . . . 29
- 10.5 Rename an existing Table . . . 30
- 10.6 List Column Families in a Table . . . 30
- 10.7 Column Family Factory . . . 30
- 10.8 Create a new Column Family . . . 30
- 10.9 Delete an existing Column Family . . . 30
- 10.10 Update an existing Column Family . . . 31

10.11 Next Step 31

11 Table **33**

12 Column Families **37**

13 Data API **41**
13.1 Cells vs. Columns vs. Column Families 41
13.2 Modifying Data 41
13.3 Reading Data 44

14 Row **47**

15 Row Data **57**

16 Google Cloud Bigtable: Python **59**
16.1 Indices and tables 59

Python Module Index **61**

To use the API, the *Client* class defines a high-level interface which handles authorization and creating other objects:

```
from gcloud_bigtable.client import Client
client = Client()
```

CHAPTER 1

Long-lived Defaults

When creating a *Client*, the `user_agent` and `timeout_seconds` arguments have sensible defaults (*DEFAULT_USER_AGENT* and *DEFAULT_TIMEOUT_SECONDS*). However, you may over-ride them and these will be used throughout all API requests made with the `client` you create.

CHAPTER 2

Authorization

This will use the Google Application Default Credentials if you don't pass any credentials of your own. If you are **familiar** with the oauth2client library, you can create a `credentials` object and pass it directly:

```
client = Client(credentials=credentials)
```

In addition, the *`from_service_account_json()`* and *`from_service_account_p12()`* factories can be used if you know the specific type of credentials you'd like to use.

CHAPTER 3

Project ID

Tip: Be sure to use the **Project ID**, not the **Project Number**.

You can also explicitly provide the `project_id` rather than relying on the inferred value:

```
client = Client(project_id='my-cloud-console-project')
```

When implicit, the value is inferred from the environment in the following order:

- The `GCLOUD_PROJECT` environment variable
- The Google App Engine application ID
- The Google Compute Engine project ID (from the metadata server)

CHAPTER 4

Admin API Access

If you'll be using your client to make Cluster Admin and Table Admin API requests, you'll need to pass the `admin` argument:

```
client = Client(admin=True)
```

CHAPTER 5

Read-Only Mode

If on the other hand, you only have (or want) read access to the data, you can pass the `read_only` argument:

```
client = Client(read_only=True)
```

This will ensure that the *READ_ONLY_SCOPE* is used for API requests (so any accidental requests that would modify data will fail).

CHAPTER 6

Next Step

After a `Client`, the next highest-level object is a `Cluster`. You'll need one before you can interact with tables or data.

Head next to learn about the Cluster Admin API.

CHAPTER 7

Client

Parent client for calling the Google Cloud Bigtable API.

This is the base from which all interactions with the API occur.

In the hierarchy of API concepts

- a *Client* owns a *Cluster*
- a *Cluster* owns a *Table*
- a *Table* owns a *ColumnFamily*
- a *Table* owns a *Row* (and all the cells in the row)

gcloud_bigtable.client.**ADMIN_SCOPE = 'https://www.googleapis.com/auth/cloud-bigtable.admin'**
Scope for interacting with the Cluster Admin and Table Admin APIs.

gcloud_bigtable.client.**CLUSTER_ADMIN_HOST = 'bigtableclusteradmin.googleapis.com'**
Cluster Admin API request host.

gcloud_bigtable.client.**CLUSTER_ADMIN_PORT = 443**
Cluster Admin API request port.

class gcloud_bigtable.client.**Client**(*credentials=None*, *project_id=None*, *read_only=False*, *admin=False*, *user_agent='gcloud-bigtable-python'*, *timeout_seconds=10*)

Bases: object

Client for interacting with Google Cloud Bigtable API.

Parameters

- **credentials** (OAuth2Credentials or NoneType) – (Optional) The OAuth2 Credentials to use for this cluster. If not provided, defaulst to the Google Application Default Credentials.
- **project_id** (str or unicode) – (Optional) The ID of the project which owns the clusters, tables and data. If not provided, will attempt to determine from the environment.
- **read_only** (*bool*) – (Optional) Boolean indicating if the data scope should be for reading only (or for writing as well). Defaults to False.
- **admin** (*bool*) – (Optional) Boolean indicating if the client will be used to interact with the Cluster Admin or Table Admin APIs. This requires the *ADMIN_SCOPE*. Defaults to False.
- **user_agent** (*str*) – (Optional) The user agent to be used with API request. Defaults to *DEFAULT_USER_AGENT*.

- **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to *DEFAULT_TIMEOUT_SECONDS*.

Raises ValueError if both read_only and admin are True

cluster(*zone*, *cluster_id*, *display_name=None*, *serve_nodes=3*)
Factory to create a cluster associated with this client.

Parameters

- **zone** (*str*) – The name of the zone where the cluster resides.
- **cluster_id** (*str*) – The ID of the cluster.
- **display_name** (*str*) – (Optional) The display name for the cluster in the Cloud Console UI. (Must be between 4 and 30 characters.) If this value is not set in the constructor, will fall back to the cluster ID.
- **serve_nodes** (*int*) – (Optional) The number of nodes in the cluster. Defaults to 3.

Return type *Cluster*

Returns The cluster owned by this client.

cluster_stub
Getter for the gRPC stub used for the Cluster Admin API.

Return type grpc.early_adopter.implementations._Stub

Returns A gRPC stub object.

Raises ValueError if the current client is not an admin client or if it has not been *start()*-ed.

credentials
Getter for client's credentials.

Return type OAuth2Credentials

Returns The credentials stored on the client.

data_stub
Getter for the gRPC stub used for the Data API.

Return type grpc.early_adopter.implementations._Stub

Returns A gRPC stub object.

Raises ValueError if the current client has not been *start()*-ed.

classmethod **from_service_account_json**(*json_credentials_path*, *project_id=None*, *read_only=False*, *admin=False*)
Factory to retrieve JSON credentials while creating client object.

Parameters

- **json_credentials_path** (*str*) – The path to a private key file (this file was given to you when you created the service account). This file must contain a JSON object with a private key and other credentials information (downloaded from the Google APIs console).
- **project_id** (*str*) – The ID of the project which owns the clusters, tables and data. Will be passed to *Client* constructor.
- **read_only** (*bool*) – Boolean indicating if the data scope should be for reading only (or for writing as well). Will be passed to *Client* constructor.

- **admin** (*bool*) – Boolean indicating if the client will be used to interact with the Cluster Admin or Table Admin APIs. Will be passed to `Client` constructor.

Return type `Client`

Returns The client created with the retrieved JSON credentials.

classmethod from_service_account_p12(*client_email*, *private_key_path*, *project_id=None*, *read_only=False*, *admin=False*)

Factory to retrieve P12 credentials while creating client object.

Note: Unless you have an explicit reason to use a PKCS12 key for your service account, we recommend using a JSON key.

Parameters

- **client_email** (*str*) – The e-mail attached to the service account.
- **private_key_path** (*str*) – The path to a private key file (this file was given to you when you created the service account). This file must be in P12 format.
- **project_id** (*str*) – The ID of the project which owns the clusters, tables and data. Will be passed to `Client` constructor.
- **read_only** (*bool*) – Boolean indicating if the data scope should be for reading only (or for writing as well). Will be passed to `Client` constructor.
- **admin** (*bool*) – Boolean indicating if the client will be used to interact with the Cluster Admin or Table Admin APIs. Will be passed to `Client` constructor.

Return type `Client`

Returns The client created with the retrieved P12 credentials.

list_clusters(*timeout_seconds=None*)

Lists clusters owned by the project.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on client.

Return type tuple

Returns A pair of results, the first is a list of `Cluster` s returned and the second is a list of strings (the failed zones in the request).

list_zones(*timeout_seconds=None*)

Lists zones associated with project.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on client.

Return type list

Returns The names (as `str`) of the zones

Raises `ValueError` if one of the zones is not in `OK` state.

operations_stub

Getter for the gRPC stub used for the Operations API.

Return type `grpc.early_adopter.implementations._Stub`

Returns A gRPC stub object.

Raises ValueError if the current client is not an admin client or if it has not been *start()*-ed.

project_id
Getter for client's project ID.

Return type str

Returns The project ID stored on the client.

project_name
Project name to be used with Cluster Admin API.

Note: This property will not change if `project_id` does not, but the return value is not cached.

The project name is of the form

```
"projects/{project_id}"
```

Return type str

Returns The project name to be used with the Cloud Bigtable Admin API RPC service.

start()
Prepare the client to make requests.

Activates gRPC contexts for making requests to the Bigtable Service(s).

stop()
Closes all the open gRPC clients.

table_stub
Getter for the gRPC stub used for the Table Admin API.

Return type `grpc.early_adopter.implementations._Stub`

Returns A gRPC stub object.

Raises ValueError if the current client is not an admin client or if it has not been *start()*-ed.

gcloud_bigtable.client.**DATA_API_HOST = 'bigtable.googleapis.com'**
Data API request host.

gcloud_bigtable.client.**DATA_API_PORT = 443**
Data API request port.

gcloud_bigtable.client.**DATA_SCOPE = 'https://www.googleapis.com/auth/cloud-bigtable.data'**
Scope for reading and writing table data.

gcloud_bigtable.client.**DEFAULT_TIMEOUT_SECONDS = 10**
The default timeout to use for API requests.

gcloud_bigtable.client.**DEFAULT_USER_AGENT = 'gcloud-bigtable-python'**
The default user agent for API requests.

gcloud_bigtable.client.**PROJECT_ENV_VAR = 'GCLOUD_PROJECT'**
Environment variable used to provide an implicit project ID.

gcloud_bigtable.client.**READ_ONLY_SCOPE = 'https://www.googleapis.com/auth/cloud-bigtable.data.readonly'**
Scope for reading table data.

gcloud_bigtable.client.**TABLE_ADMIN_HOST = 'bigtabletableadmin.googleapis.com'**
Table Admin API request host.

gcloud_bigtable.client.**TABLE_ADMIN_PORT = 443**
Table Admin API request port.

CHAPTER 8

Cluster Admin API

After creating a *Client*, you can interact with individual clusters, groups of clusters or available zones for a project.

8.1 List Clusters

If you want a comprehensive list of all existing clusters, make a ListClusters API request with *Client.list_clusters()*:

```
clusters = client.list_clusters()
```

8.2 List Zones

If you aren't sure which `zone` to create a cluster in, find out which zones your project has access to with a ListZones API request with *Client.list_zones()*:

```
zones = client.list_zones()
```

You can choose a `string` from among the result to pass to the *Cluster* constructor.

8.3 Cluster Factory

To create a *Cluster* object:

```
cluster = client.cluster(zone, cluster_id,
                         display_name=display_name,
                         serve_nodes=3)
```

Both `display_name` and `serve_nodes` are optional. When not provided, `display_name` defaults to the `cluster_id` value and `serve_nodes` defaults to the minimum allowed: `3`.

Even if this *Cluster* already has been created with the API, you'll want this object to use as a parent of a *Table* just as the *Client* is used as the parent of a *Cluster*.

8.4 Create a new Cluster

After creating the cluster object, make a CreateCluster API request with *create()*:

```
cluster.display_name = 'My very own cluster'
cluster.create()
```

If you would like more than the minimum number of nodes (3) in your cluster:

```
cluster.serve_nodes = 10
cluster.create()
```

8.5 Check on Current Operation

Note: When modifying a cluster (via a CreateCluster, UpdateCluster or UndeleteCluster request), the Bigtable API will return a long-running Operation. This will be stored on the object after each of *create()*, *update()* and *undelete()* are called.

You can check if a long-running operation (for a *create()*, *update()* or *undelete()*) has finished by making a GetOperation request with *operation_finished()*:

```
>>> cluster.operation_finished()
True
```

Note: The operation data is stored in protected fields on the *Cluster*: _operation_type, _operation_id and _operation_begin. If these are unset, then *operation_finished()* will fail. Also, these will be removed after a long-running operation has completed (checked via this method). We could easily surface these properties publicly, but it's unclear if end-users would need them.

8.6 Get metadata for an existing Cluster

After creating the cluster object, make a GetCluster API request with *reload()*:

```
cluster.reload()
```

This will load serve_nodes and display_name for the existing cluster in addition to the cluster_id, zone and project_id already set on the *Cluster* object.

8.7 Update an existing Cluster

After creating the cluster object, make an UpdateCluster API request with *update()*:

```
client.display_name = 'New display_name'
cluster.update()
```

8.8 Delete an existing Cluster

Make a DeleteCluster API request with *delete()*:

```
cluster.delete()
```

8.9 Undelete a deleted Cluster

Make an UndeleteCluster API request with *undelete()*:

```
cluster.undelete()
```

8.10 Next Step

Now we go down the hierarchy from *Cluster* to a *Table*.

Head next to learn about the Table Admin API.

CHAPTER 9

Cluster

User friendly container for Google Cloud Bigtable Cluster.

class gcloud_bigtable.cluster.**Cluster**(*zone*, *cluster_id*, *client*, *display_name=None*, *serve_nodes=3*)

Bases: object

Representation of a Google Cloud Bigtable Cluster.

We can use a *Cluster* to:

- *reload()* itself
- *create()* itself
- Check if an *operation_finished()* (each of *create()*, *update()* and *undelete()* return with long-running operations)
- *update()* itself
- *delete()* itself
- *undelete()* itself

Note: For now, we leave out the properties hdd_bytes and ssd_bytes (both integers) and also the default_storage_type (an enum) which if not sent will end up as data_pb2.STORAGE_SSD.

Parameters

- **zone** (*str*) – The name of the zone where the cluster resides.
- **cluster_id** (*str*) – The ID of the cluster.
- **client** (*client.Client*) – The client that owns the cluster. Provides authorization and a project ID.
- **display_name** (*str*) – (Optional) The display name for the cluster in the Cloud Console UI. (Must be between 4 and 30 characters.) If this value is not set in the constructor, will fall back to the cluster ID.
- **serve_nodes** (*int*) – (Optional) The number of nodes in the cluster. Defaults to 3.

client

Getter for cluster's client.

Return type *client.Client*

Returns The client stored on the cluster.

create(*timeout_seconds=None*)
Create this cluster.

Note: Uses the `project_id`, `zone` and `cluster_id` on the current *Cluster* in addition to the `display_name` and `serve_nodes`. If you'd like to change them before creating, reset the values via

```
cluster.display_name = 'New display name'
cluster.cluster_id = 'i-changed-my-mind'
```

before calling *create()*.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

delete(*timeout_seconds=None*)
Delete this cluster.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

classmethod from_pb(*cluster_pb*, *client*)
Creates a cluster instance from a protobuf.

Parameters

- **cluster_pb** (`bigtable_cluster_data_pb2.Cluster`) – A cluster protobuf object.
- **client** (*client.Client*) – The client that owns the cluster.

Return type *Cluster*

Returns The cluster parsed from the protobuf response.

Raises `ValueError` if the cluster name does not match `_CLUSTER_NAME_RE` or if the parsed project ID does not match the project ID on the client.

list_tables(*timeout_seconds=None*)
List the tables in this cluster.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

Return type list of *Table*

Returns The list of tables owned by the cluster.

Raises `ValueError` if one of the returned tables has a name that is not of the expected format.

name
Cluster name used in requests.

Note: This property will not change if `zone` and `cluster_id` do not, but the return value is not cached.

The cluster name is of the form

```
"projects/{project_id}/zones/{zone}/clusters/{cluster_id}"
```

Return type str

Returns The cluster name.

operation_finished(*timeout_seconds=None*)
Check if the current operation has finished.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

Return type bool

Returns A boolean indicating if the current operation has completed.

Raises `ValueError` if there is no current operation set.

project_id
Getter for cluster's project ID.

Return type str

Returns The project ID for the cluster (is stored on the client).

reload(*timeout_seconds=None*)
Reload the metadata for this cluster.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

table(*table_id*)
Factory to create a table associated with this cluster.

Parameters **table_id** (*str*) – The ID of the table.

Return type *Table*

Returns The table owned by this cluster.

timeout_seconds
Getter for cluster's default timeout seconds.

Return type int

Returns The timeout seconds default stored on the cluster's client.

undelete(*timeout_seconds=None*)
Undelete this cluster.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

update(*timeout_seconds=None*)
Update this cluster.

Note: Updates the `display_name` and `serve_nodes`. If you'd like to change them before updating, reset the values via

```
cluster.display_name = 'New display name'
cluster.serve_nodes = 3
```

before calling *update()*.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on cluster.

CHAPTER 10

Table Admin API

After creating a *Cluster*, you can interact with individual tables, groups of tables or column families within a table.

10.1 List Tables

If you want a comprehensive list of all existing tables in a cluster, make a ListTables API request with *Cluster.list_tables()*:

```
tables = cluster.list_tables()
```

10.2 Table Factory

To create a *Table* object:

```
table = cluster.table(table_id)
```

Even if this *Table* already has been created with the API, you'll want this object to use as a parent of a *ColumnFamily* or *Row*.

10.3 Create a new Table

After creating the table object, make a CreateTable API request with *create()*:

```
table.create()
```

If you would to initially split the table into several tablets (Tablets are similar to HBase regions):

```
table.create(initial_split_keys=['s1', 's2'])
```

10.4 Delete an existing Table

Make a DeleteTable API request with *delete()*:

```
table.delete()
```

10.5 Rename an existing Table

Though the RenameTable API request is listed in the service definition, requests to that method return:

```
BigtableTableService.RenameTable is not yet implemented
```

We have implemented *rename()* but it will not work unless the backend supports the method.

10.6 List Column Families in a Table

Though there is no **official** method for retrieving column families associated with a table, the GetTable API method returns a table object with the names of the column families.

To retrieve the list of column families use *list_column_families()*:

```
column_families = table.list_column_families()
```

Note: Unfortunately the garbage collection rules used to create each column family are not returned in the GetTable response.

10.7 Column Family Factory

To create a *ColumnFamily* object:

```
column_family = table.column_family(column_family_id)
```

There is no real reason to use this factory unless you intend to create or delete a column family.

In addition, you can specify an optional `gc_rule` (a *GarbageCollectionRule* or similar):

```
column_family = table.column_family(column_family_id,
                                    gc_rule=gc_rule)
```

This rule helps the backend determine when and how to clean up old cells in the column family.

See the Column Families doc for more information about *GarbageCollectionRule* and related classes.

10.8 Create a new Column Family

After creating the column family object, make a CreateColumnFamily API request with *ColumnFamily.create()*

```
column_family.create()
```

10.9 Delete an existing Column Family

Make a DeleteColumnFamily API request with *ColumnFamily.delete()*

```
column_family.delete()
```

10.10 Update an existing Column Family

Though the UpdateColumnFamily API request is listed in the service definition, requests to that method return:

```
BigtableTableService.UpdateColumnFamily is not yet implemented
```

We have implemented *ColumnFamily.update()* but it will not work unless the backend supports the method.

10.11 Next Step

Now we go down the final step of the hierarchy from *Table* to *Row* as well as streaming data directly via a *Table*.

Head next to learn about the Data API.

CHAPTER 11

Table

User friendly container for Google Cloud Bigtable Table.

class gcloud_bigtable.table.**Table**(*table_id*, *cluster*)
Bases: object

Representation of a Google Cloud Bigtable Table.

Note: We don't define any properties on a table other than the name. As the proto says, in a request:

> The name field of the Table and all of its ColumnFamilies must be left blank, and will be populated in the response.

This leaves only the current_operation and granularity fields. The current_operation is only used for responses while granularity is an enum with only one value.

We can use a *Table* to:

- *create()* the table
- *rename()* the table
- *delete()* the table
- *list_column_families()* in the table

Parameters

- **table_id** (*str*) – The ID of the table.
- **cluster** (*cluster.Cluster*) – The cluster that owns the table.

client
Getter for table's client.

Return type *client.Client*

Returns The client that owns this table.

cluster
Getter for table's cluster.

Return type *cluster.Cluster*

Returns The cluster stored on the table.

column_family(*column_family_id*, *gc_rule=None*)
Factory to create a column family associated with this table.

Parameters

- **column_family_id** (*str*) – The ID of the column family. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **gc_rule** (*column_family.GarbageCollectionRule*, *column_family.GarbageCollectionRuleUnion* or *column_family.GarbageCollectionRuleIntersection*) – (Optional) The garbage collection settings for this column family.

Return type *column_family.ColumnFamily*

Returns A column family owned by this table.

create(*initial_split_keys=None*, *timeout_seconds=None*)
Creates this table.

Note: Though a `_generated.bigtable_table_data_pb2.Table` is also allowed (as the `table` property) in a create table request, we do not support it in this method. As mentioned in the *Table* docstring, the name is the only useful property in the table proto.

Note: A create request returns a `_generated.bigtable_table_data_pb2.Table` but we don't use this response. The proto definition allows for the inclusion of a `current_operation` in the response, but in example usage so far, it seems the Bigtable API does not return any operation.

Parameters

- **initial_split_keys** (*list*) – (Optional) List of row keys that will be used to initially split the table into several tablets (Tablets are similar to HBase regions). Given two split keys, `"s1"` and `"s2"`, three tablets will be created, spanning the key ranges: `[, s1)`, `[s1, s2)`, `[s2, )`.
- **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

delete(*timeout_seconds=None*)
Delete this table.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

list_column_families(*timeout_seconds=None*)
Check if this table exists.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

Return type dictionary with string as keys and *column_family.ColumnFamily* as values

Returns List of column families attached to this table.

Raises `ValueError` if the column family name from the response does not agree with the computed name from the column family ID.

name
Table name used in requests.

Note: This property will not change if `table_id` does not, but the return value is not cached.

The table name is of the form

```
"projects/../zones/../clusters/../tables/{table_id}"
```

Return type str

Returns The table name.

read_row(*row_key*, *filter=None*, *timeout_seconds=None*)

Read a single row from this table.

Parameters

- **row_key** (*bytes*) – The key of the row to read from.
- **filter** (*row.RowFilter*, *row.RowFilterChain*, *row.RowFilterUnion* or *row.ConditionalRowFilter*) – (Optional) The filter to apply to the contents of the row. If unset, returns the entire row.
- **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

Return type *PartialRowData*

Returns The contents of the row.

Raises ValueError if a commit row chunk is never encountered.

read_rows(*start_key=None*, *end_key=None*, *allow_row_interleaving=None*, *limit=None*, *filter=None*, *timeout_seconds=None*)

Read rows from this table.

Parameters

- **start_key** (*bytes*) – (Optional) The beginning of a range of row keys to read from. The range will include start_key. If left empty, will be interpreted as the empty string.
- **end_key** (*bytes*) – (Optional) The end of a range of row keys to read from. The range will not include end_key. If left empty, will be interpreted as an infinite string.
- **filter** (*row.RowFilter*, *row.RowFilterChain*, *row.RowFilterUnion* or *row.ConditionalRowFilter*) – (Optional) The filter to apply to the contents of the specified row(s). If unset, reads every column in each row.
- **allow_row_interleaving** (*bool*) – (Optional) By default, rows are read sequentially, producing results which are guaranteed to arrive in increasing row order. Setting allow_row_interleaving to True allows multiple rows to be interleaved in the response stream, which increases throughput but breaks this guarantee, and may force the client to use more memory to buffer partially-received rows.
- **limit** (*int*) – (Optional) The read will terminate after committing to N rows' worth of results. The default (zero) is to return all results. Note that if allow_row_interleaving is set to True, partial results may be returned for more than N rows. However, only N commit_row chunks will be sent.
- **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

Return type *PartialRowsData*

Returns A *PartialRowsData* convenience wrapper for consuming the streamed results.

rename(*new_table_id*, *timeout_seconds=None*)
Rename this table.

Note: This cannot be used to move tables between clusters, zones, or projects.

Note: The Bigtable Table Admin API currently returns

```
BigtableTableService.RenameTable is not yet implemented
```

when this method is used. It's unclear when this method will actually be supported by the API.

Parameters

- **new_table_id** (*str*) – The new name table ID.
- **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

row(*row_key*)
Factory to create a row associated with this table.

Parameters **row_key** (*bytes*) – The key for the row being created.

Return type *row.Row*

Returns A row owned by this table.

sample_row_keys(*timeout_seconds=None*)
Read a sample of row keys in the table.

The returned row keys will delimit contiguous sections of the table of approximately equal size, which can be used to break up the data for distributed tasks like mapreduces.

The elements in the iterator are a SampleRowKeys response and they have the properties `offset_bytes` and `row_key`. They occur in sorted order. The table might have contents before the first row key in the list and after the last one, but a key containing the empty string indicates "end of table" and will be the last response given, if present.

Note: Row keys in this list may not have ever been written to or read from, and users should therefore not make any assumptions about the row key structure that are specific to their use case.

The `offset_bytes` field on a response indicates the approximate total storage space used by all rows in the table which precede `row_key`. Buffering the contents of all rows between two subsequent samples would require space roughly equal to the difference in their `offset_bytes` fields.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on table.

Return type `grpc.framework.alpha._reexport._CancellableIterator`

Returns A cancel-able iterator. Can be consumed by calling `next()` or by casting to a `list` and can be cancelled by calling `cancel()`.

timeout_seconds
Getter for table's default timeout seconds.

Return type int

Returns The timeout seconds default stored on the table's client.

CHAPTER 12

Column Families

When creating a *Table*, it is possible to set garbage collection rules for expired data.

By setting a rule, cells in the table matching the rule will be deleted during periodic garbage collection (which executes opportunistically in the background).

The types *GarbageCollectionRule*, *GarbageCollectionRuleUnion* and *GarbageCollectionRuleIntersection* can all be used as the optional `gc_rule` argument in the *ColumnFamily* constructor. This value is then used in the *create()* and *update()* methods.

These rules can be nested arbitrarily, with *GarbageCollectionRule* at the lowest level of the nesting:

```
import datetime

max_age = datetime.timedelta(days=3)
rule1 = GarbageCollectionRule(max_age=max_age)
rule2 = GarbageCollectionRule(max_num_versions=1)

# Make a composite that matches anything older than 3 days **AND**
# with more than 1 version.
rule3 = GarbageCollectionIntersection(rules=[rule1, rule2])

# Make another composite that matches our previous intersection
# **OR** anything that has more than 3 versions.
rule4 = GarbageCollectionRule(max_num_versions=3)
rule5 = GarbageCollectionUnion(rules=[rule3, rule4])
```

User friendly container for Google Cloud Bigtable Column Family.

class `gcloud_bigtable.column_family.`**`ColumnFamily`**(*column_family_id*, *table*, *gc_rule=None*)

Bases: `object`

Representation of a Google Cloud Bigtable Column Family.

We can use a *ColumnFamily* to:

- *create()* itself
- *update()* itself
- *delete()* itself

Parameters

- **column_family_id** (*str*) – The ID of the column family. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **table** (*table.Table*) – The table that owns the column family.
- **gc_rule** (*GarbageCollectionRule*, *GarbageCollectionRuleUnion* or *GarbageCollectionRuleIntersection*) – (Optional) The garbage collection settings for this column family.

client
Getter for column family's client.

Return type *client.Client*

Returns The client that owns this column family.

create(*timeout_seconds=None*)
Create this column family.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on column family.

delete(*timeout_seconds=None*)
Delete this column family.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on column family.

name
Column family name used in requests.

Note: This property will not change if `column_family_id` does not, but the return value is not cached.

The table name is of the form

```
"projects/../zones/../clusters/../tables/../columnFamilies/.."
```

Return type str

Returns The column family name.

table
Getter for column family's table.

Return type *table.Table*

Returns The table stored on the column family.

timeout_seconds
Getter for column family's default timeout seconds.

Return type int

Returns The timeout seconds default.

update(*timeout_seconds=None*)
Update this column family.

Note: The Bigtable Table Admin API currently returns

```
BigtableTableService.UpdateColumnFamily is not yet implemented
```

when this method is used. It's unclear when this method will actually be supported by the API.

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on column family.

class gcloud_bigtable.column_family.**GarbageCollectionRule**(*max_num_versions=None*, *max_age=None*)

Bases: object

Table garbage collection rule.

Cells in the table fitting the rule will be deleted during garbage collection.

These values can be combined via *GarbageCollectionRuleUnion* and *GarbageCollectionRuleIntersection*.

Note: At most one of max_num_versions and max_age can be specified at once.

Note: A string gc_expression can also be used with API requests, but that value would be superceded by a gc_rule. As a result, we don't support that feature and instead support via this native object.

Parameters

- **max_num_versions** (*int*) – The maximum number of versions
- **max_age** (datetime.timedelta) – The maximum age allowed for a cell in the table.

Raises TypeError if both max_num_versions and max_age are set.

to_pb()

Converts the *GarbageCollectionRule* to a protobuf.

Return type data_pb2.GcRule

Returns The converted current object.

class gcloud_bigtable.column_family.**GarbageCollectionRuleIntersection**(*rules=None*)

Bases: object

Intersection of garbage collection rules.

Parameters **rules** (*list*) – List of *GarbageCollectionRule*, *GarbageCollectionRuleUnion* and/or *GarbageCollectionRuleIntersection*

to_pb()

Converts the intersection into a single gc rule as a protobuf.

Return type data_pb2.GcRule

Returns The converted current object.

class gcloud_bigtable.column_family.**GarbageCollectionRuleUnion**(*rules=None*)

Bases: object

Union of garbage collection rules.

Parameters **rules** (*list*) – List of *GarbageCollectionRule*, *GarbageCollectionRuleUnion* and/or *GarbageCollectionRuleIntersection*

to_pb()

Converts the union into a single gc rule as a protobuf.

Return type `data_pb2.GcRule`

Returns The converted current object.

CHAPTER 13

Data API

After creating a *Table* and some column families, you are ready to store and retrieve data.

13.1 Cells vs. Columns vs. Column Families

- As we saw before, a table can have many column families.
- As we'll see below, a table also has many rows (specified by row keys).
- Within a row, data is stored in a cell. A cell simply has a value (as bytes) and a timestamp. The number of cells in each row can be different, depending on what was stored in each row.
- Each cell lies in a column (**not** a column family). A column is really just a more **specific** modifier within a column family. A column can be present in every way, in only one or anywhere in between.
- Within a column family there can be many columns. For example within the column family `foo` we could have columns `bar` and `baz`. These would typically be represented as `foo:bar` and `foo:baz`.

13.2 Modifying Data

Since data is stored in cells, which are stored in rows, the *Row* class is the only class used to modify (write, update, delete) data in a *Table*.

13.2.1 Row Factory

To create a *Row* object

```
row = table.row(row_key)
```

Unlike the previous string values we've used before, the row key must be `bytes`.

13.2.2 Direct vs. Conditional vs. Append

There are three ways to modify data in a table, described by the MutateRow, CheckAndMutateRow and ReadModifyWriteRow API methods.

- The **direct** way is via MutateRow which involves simply adding, overwriting or deleting cells.

- The **conditional** way is via CheckAndMutateRow. This method first checks if some filter is matched in a a given row, then applies one of two sets of mutations, depending on if a match occurred or not.
- The **append** way is via ReadModifyWriteRow. This simply appends (as bytes) or increments (as an integer) data in a presumed existing cell in a row.

13.2.3 Building Up Mutations

In all three cases, a set of mutations (or two sets) are built up on a *Row* before they are sent of in a batch via *commit()*:

```
row.commit()
```

To send **append** mutations in batch, use *commit_modifications()*:

```
row.commit_modifications()
```

We have a small set of methods on the *Row* to build these mutations up.

13.2.4 Direct Mutations

Direct mutations can be added via one of four methods

- *set_cell()* allows a single value to be written to a column

```
row.set_cell(column_family_id, column, value,
             timestamp=timestamp)
```

  If the `timestamp` is omitted, the current time on the Google Cloud Bigtable server will be used when the cell is stored.

  The value can either by bytes or an integer (which will be converted to bytes as an unsigned 64-bit integer).

- *delete_cell()* deletes all cells (i.e. for all timestamps) in a given column

```
row.delete_cell(column_family_id, column)
```

  Remember, this only happens in the `row` we are using.

  If we only want to delete cells from a limited range of time, a *TimestampRange* can be used

```
row.delete_cell(column_family_id, column,
                time_range=time_range)
```

- *delete_cells()* does the same thing as *delete_cell()* but accepts a list of columns in a column family rather than a single one.

```
row.delete_cells(column_family_id, [column1, column2],
                 time_range=time_range)
```

  In addition, if we want to delete cells from every column in a column family, the special *ALL_COLUMNS* value can be used

```
row.delete_cells(column_family_id, Row.ALL_COLUMNS,
                 time_range=time_range)
```

- *delete()* will delete the entire row

```
row.delete()
```

13.2.5 Conditional Mutations

Making **conditional** conditional modifications is essentially identical to **direct** modifications, but we need to specify a filter to match against in the row:

```
row = table.row(row_key, filter=filter)
```

See the *Row* class for more information about acceptable values for `filter`.

The only other difference from **direct** modifications are that each mutation added must specify a `state`: will the mutation be applied if the filter matches or if it fails to match.

For example

```
row.set_cell(column_family_id, column, value,
             timestamp=timestamp, state=True)
```

Note: If `state` is passed when no `filter` is set on a *Row*, adding the mutation will fail. Similarly, if no `state` is passed when a `filter` has been set, adding the mutation will fail.

13.2.6 Append Mutations

Append mutations can be added via one of two methods

- *append_cell_value* appends a bytes value to an existing cell:

```
row.append_cell_value(column_family_id, column, bytes_value)
```

- *increment_cell_value* increments an integer value in an existing cell:

```
row.increment_cell_value(column_family_id, column, int_value)
```

  Since only bytes are stored in a cell, the current value is decoded as an unsigned 64-bit integer before being incremented. (This happens on the Google Cloud Bigtable server, not in the library.)

Notice that no timestamp was specified. This is because **append** mutations operate on the latest value of the specified column.

If there are no cells in the specified column, then the empty string (bytes case) or zero (integer case) are the assumed values.

13.2.7 Starting Fresh

If accumulated mutations need to be dropped, use *clear_mutations()*

```
row.clear_mutations()
```

To clear **append** mutations, use *clear_modification_rules()*

```
row.clear_modification_rules()
```

13.3 Reading Data

13.3.1 Read Single Row from a Table

To make a ReadRows API request for a single row key, use *Table.read_row()*:

```
row_data = table.read_row(row_key)
```

Rather than returning a *Row*, this method returns a *PartialRowData* instance. This class is used for reading and parsing data rather than for modifying data (as *Row* is).

A filter can also be applied to the

```
row_data = table.read_row(row_key, filter=filter)
```

The allowable `filter` values are the same as those used for a *Row* with **conditional** mutations. For more information, see the *Table.read_row()* documentation.

13.3.2 Stream Many Rows from a Table

To make a ReadRows API request for a stream of rows, use *Table.read_rows()*:

```
row_data = table.read_rows()
```

Using gRPC over HTTP/2, a continual stream of responses will be delivered. We have a custom returns a *PartialRowsData* class to allow consuming and parsing these streams as they come.

In particular

- *consume_next()* pulls the next result from the stream, parses it and stores it on the *PartialRowsData* instance
- *consume_all()* pulls results from the stream until there are no more
- *cancel()* closes the stream

See the *PartialRowsData* documentation for more information.

As with *Table.read_row()*, an optional `filter` can be applied. In addition a `start_key` and / or `end_key` can be supplied for the stream, a `limit` can be set and a boolean `allow_row_interleaving` can be specified to allow faster streamed results at the potential cost of non-sequential reads.

See the *Table.read_rows()* documentation for more information on the optional arguments.

13.3.3 Sample Keys in a Table

Make a SampleRowKeys API request with *Table.sample_row_keys()*:

```
keys_iterator = table.sample_row_keys()
```

The returned row keys will delimit contiguous sections of the table of approximately equal size, which can be used to break up the data for distributed tasks like mapreduces.

As with *Table.read_rows()*, the returned `keys_iterator` is connected to a cancellable HTTP/2 stream.

The next key in the result can be accessed via

```
next_key = keys_iterator.next()
```

or all keys can be iterated over via

```
for curr_key in keys_iterator:
    do_something(curr_key)
```

Just as with reading, the stream can be canceled:

```
keys_iterator.cancel()
```

CHAPTER 14

Row

User friendly container for Google Cloud Bigtable Row.

class gcloud_bigtable.row.**CellValueRange**(*start_value=None*, *end_value=None*, *inclusive_start=True*, *inclusive_end=True*)

Bases: object

A range of values to restrict to in a row filter.

With only match cells that have values in this range.

Both the start and end value can be included or excluded in the range. By default, we include them both, but this can be changed with optional flags.

Parameters

- **start_value** (*bytes*) – The start of the range of values. If no value is used, it is interpreted as the empty string (inclusive) by the backend.
- **end_value** (*bytes*) – The end of the range of values. If no value is used, it is interpreted as the infinite string (exclusive) by the backend.
- **inclusive_start** (*bool*) – Boolean indicating if the start value should be included in the range (or excluded).
- **inclusive_end** (*bool*) – Boolean indicating if the end value should be included in the range (or excluded).

to_pb()

Converts the *CellValueRange* to a protobuf.

Return type data_pb2.ValueRange

Returns The converted current object.

class gcloud_bigtable.row.**ColumnRange**(*column_family_id*, *start_column=None*, *end_column=None*, *inclusive_start=True*, *inclusive_end=True*)

Bases: object

A range of columns to restrict to in a row filter.

Both the start and end column can be included or excluded in the range. By default, we include them both, but this can be changed with optional flags.

Parameters

- **column_family_id** (*str*) – The column family that contains the columns. Must be of the form [_a-zA-Z0-9][-_.a-zA-Z0-9]*.

- **start_column** (*bytes*) – The start of the range of columns. If no value is used, it is interpreted as the empty string (inclusive) by the backend.
- **end_column** (*bytes*) – The end of the range of columns. If no value is used, it is interpreted as the infinite string (exclusive) by the backend.
- **inclusive_start** (*bool*) – Boolean indicating if the start column should be included in the range (or excluded).
- **inclusive_end** (*bool*) – Boolean indicating if the end column should be included in the range (or excluded).

to_pb()
Converts the *ColumnRange* to a protobuf.

Return type data_pb2.ColumnRange

Returns The converted current object.

class gcloud_bigtable.row.**ConditionalRowFilter**(*base_filter*, *true_filter=None*, *false_filter=None*)
Bases: object

Conditional filter

Executes one of two filters based on another filter. If the base_filter returns any cells in the row, then true_filter is executed. If not, then false_filter is executed.

Note: The base_filter does not execute atomically with the true and false filters, which may lead to inconsistent or unexpected results.

Additionally, executing a *ConditionalRowFilter* has poor performance on the server, especially when false_filter is set.

Parameters

- **base_filter** (*RowFilter*, *RowFilterChain*, *RowFilterUnion* or *ConditionalRowFilter*) – The filter to condition on before executing the true/false filters.
- **true_filter** (*RowFilter*, *RowFilterChain*, *RowFilterUnion* or *ConditionalRowFilter*) – (Optional) The filter to execute if there are any cells matching base_filter. If not provided, no results will be returned in the true case.
- **false_filter** (*RowFilter*, *RowFilterChain*, *RowFilterUnion* or *ConditionalRowFilter*) – (Optional) The filter to execute if there are no cells matching base_filter. If not provided, no results will be returned in the false case.

to_pb()
Converts the *ConditionalRowFilter* to a protobuf.

Return type data_pb2.RowFilter

Returns The converted current object.

class gcloud_bigtable.row.**Row**(*row_key*, *table*, *filter=None*)
Bases: object

Representation of a Google Cloud Bigtable Row.

Note: A *Row* accumulates mutations locally via the *set_cell()*, *delete()*, *delete_cell()* and *delete_cells()* methods. To actually send these mutations to the Google Cloud Bigtable API, you must

call *commit()*. If a `filter` is set on the *Row*, the mutations must have an associated state: `True` or `False`. The mutations will be applied conditionally, based on whether the filter matches any cells in the *Row* or not.

Parameters

- **row_key** (*bytes*) – The key for the current row.
- **table** (*table.Table*) – The table that owns the row.
- **filter** (*RowFilter*, *RowFilterChain*, *RowFilterUnion* or *ConditionalRowFilter*) – (Optional) Filter to be used for conditional mutations. If a filter is set, then the *Row* will accumulate mutations for either a `True` or `False` state. When *commit()*-ed, the mutations for the `True` state will be applied if the filter matches any cells in the row, otherwise the `False` state will be.

ALL_COLUMNS = <object object>
Sentinel value used to indicate all columns in a column family.

append_cell_value(*column_family_id*, *column*, *value*)
Appends a value to an existing cell.

Note: This method adds a read-modify rule protobuf to the accumulated read-modify rules on this *Row*, but does not make an API request. To actually send an API request (with the rules) to the Google Cloud Bigtable API, call *commit()*.

Parameters

- **column_family_id** (*str*) – The column family that contains the column. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **column** (*bytes*) – The column within the column family where the cell is located.
- **value** (*bytes*) – The value to append to the existing value in the cell. If the targeted cell is unset, it will be treated as containing the empty string.

clear_modification_rules()
Removes all currently accumulated modifications on current row.

clear_mutations()
Removes all currently accumulated mutations on the current row.

client
Getter for row's client.

Return type *client.Client*

Returns The client that owns this row.

commit(*timeout_seconds=None*)
Makes a `MutateRow` or `CheckAndMutateRow` API request.

If no mutations have been created in the row, no request is made.

Mutations are applied atomically and in order, meaning that earlier mutations can be masked / negated by later ones. Cells already present in the row are left unchanged unless explicitly changed by a mutation.

After committing the accumulated mutations, resets the local mutations to an empty list.

In the case that a filter is set on the *Row*, the mutations will be applied conditionally, based on whether the filter matches any cells in the *Row* or not. (Each method which adds a mutation has a `state` parameter for this purpose.)

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on row.

Return type `bool` or `NoneType`

Returns `None` if there is no filter, otherwise a flag indicating if the filter was matched (which also indicates which set of mutations were applied by the server).

Raises `ValueError` if the number of mutations exceeds the `_MAX_MUTATIONS`.

commit_modifications(*timeout_seconds=None*)

Makes a `ReadModifyWriteRow` API request.

This commits modifications made by *append_cell_value()* and *increment_cell_value()*. If no modifications were made, makes no API request and just returns `{}`.

Modifies a row atomically, reading the latest existing timestamp/value from the specified columns and writing a new value by appending / incrementing. The new cell created uses either the current server time or the highest timestamp of a cell in that column (if it exceeds the server time).

Parameters **timeout_seconds** (*int*) – Number of seconds for request time-out. If not passed, defaults to value set on row.

Return type dict

Returns

The new contents of all modified cells. Returned as a dictionary of column families, each of which holds a dictionary of columns. Each column contains a list of cells modified. Each cell is represented with a two-tuple with the value (in bytes) and the timestamp for the cell. For example:

```
{
    u'col-fam-id': {
        b'col-name1': [
            (b'cell-val', datetime.datetime(...)),
            (b'cell-val-newer', datetime.datetime(...)),
        ],
        b'col-name2': [
            (b'altcol-cell-val', datetime.datetime(...)),
        ],
    },
    u'col-fam-id2': {
        b'col-name3-but-other-fam': [
            (b'foo', datetime.datetime(...)),
        ],
    },
}
```

delete(*state=None*)

Deletes this row from the table.

Note: This method adds a mutation to the accumulated mutations on this *Row*, but does not make an API request. To actually send an API request (with the mutations) to the Google Cloud Bigtable API, call *commit()*.

Parameters **state** (*bool*) – (Optional) The state that the mutation should be applied in. Unset if the mutation is not conditional, otherwise `True` or `False`.

delete_cell(*column_family_id*, *column*, *time_range=None*, *state=None*)
Deletes cell in this row.

Note: This method adds a mutation to the accumulated mutations on this *Row*, but does not make an API request. To actually send an API request (with the mutations) to the Google Cloud Bigtable API, call *commit()*.

Parameters

- **column_family_id** (*str*) – The column family that contains the column or columns with cells being deleted. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **column** (*bytes*) – The column within the column family that will have a cell deleted.
- **time_range** (*TimestampRange*) – (Optional) The range of time within which cells should be deleted.
- **state** (*bool*) – (Optional) The state that the mutation should be applied in. Unset if the mutation is not conditional, otherwise `True` or `False`.

delete_cells(*column_family_id*, *columns*, *time_range=None*, *state=None*)
Deletes cells in this row.

Note: This method adds a mutation to the accumulated mutations on this *Row*, but does not make an API request. To actually send an API request (with the mutations) to the Google Cloud Bigtable API, call *commit()*.

Parameters

- **column_family_id** (*str*) – The column family that contains the column or columns with cells being deleted. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **columns** (`list` of `str` / `unicode`, or `object`) – The columns within the column family that will have cells deleted. If *Row.ALL_COLUMNS* is used then the entire column family will be deleted from the row.
- **time_range** (*TimestampRange*) – (Optional) The range of time within which cells should be deleted.
- **state** (*bool*) – (Optional) The state that the mutation should be applied in. Unset if the mutation is not conditional, otherwise `True` or `False`.

filter
Getter for row's filter.

Return type *RowFilter*, *RowFilterChain*, *RowFilterUnion*, *ConditionalRowFilter* or `NoneType`

Returns The filter for the row.

increment_cell_value(*column_family_id*, *column*, *int_value*)
Increments a value in an existing cell.

Assumes the value in the cell is stored as a 64 bit integer serialized to bytes.

Note: This method adds a read-modify rule protobuf to the accumulated read-modify rules on this *Row*,

but does not make an API request. To actually send an API request (with the rules) to the Google Cloud Bigtable API, call *commit()*.

Parameters

- **column_family_id** (*str*) – The column family that contains the column. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **column** (*bytes*) – The column within the column family where the cell is located.
- **int_value** (*int*) – The value to increment the existing value in the cell by. If the targeted cell is unset, it will be treated as containing a zero. Otherwise, the targeted cell must contain an 8-byte value (interpreted as a 64-bit big-endian signed integer), or the entire request will fail.

row_key
Getter for row's key.

Return type bytes

Returns The key for the row.

set_cell(*column_family_id*, *column*, *value*, *timestamp=None*, *state=None*)
Sets a value in this row.

The cell is determined by the `row_key` of the *Row* and the `column`. The `column` must be in an existing *column_family.ColumnFamily* (as determined by `column_family_id`).

Note: This method adds a mutation to the accumulated mutations on this *Row*, but does not make an API request. To actually send an API request (with the mutations) to the Google Cloud Bigtable API, call *commit()*.

Parameters

- **column_family_id** (*str*) – The column family that contains the column. Must be of the form `[_a-zA-Z0-9][-_.a-zA-Z0-9]*`.
- **column** (*bytes*) – The column within the column family where the cell is located.
- **value** (bytes or `int`) – The value to set in the cell. If an integer is used, will be interpreted as a 64-bit big-endian signed integer (8 bytes).
- **timestamp** (`datetime.datetime`) – (Optional) The timestamp of the operation.
- **state** (*bool*) – (Optional) The state that the mutation should be applied in. Unset if the mutation is not conditional, otherwise `True` or `False`.

table
Getter for row's table.

Return type *table.Table*

Returns The table stored on the row.

timeout_seconds
Getter for row's default timeout seconds.

Return type int

Returns The timeout seconds default.

class gcloud_bigtable.row.**RowFilter**(*row_key_regex_filter=None*, *family_name_regex_filter=None*, *column_qualifier_regex_filter=None*, *value_regex_filter=None*, *column_range_filter=None*, *timestamp_range_filter=None*, *value_range_filter=None*, *cells_per_row_offset_filter=None*, *cells_per_row_limit_filter=None*, *cells_per_column_limit_filter=None*, *row_sample_filter=None*, *strip_value_transformer=None*)

Bases: object

Basic filter to apply to cells in a row.

These values can be combined via *RowFilterChain*, *RowFilterUnion* and *ConditionalRowFilter*.

The regex filters must be valid RE2 patterns. See Google's RE2 reference for the accepted syntax.

Note: At most one of the keyword arguments can be specified at once.

Note: For bytes regex filters (row_key, column_qualifier and value), special care need be used with the expression used. Since each of these properties can contain arbitrary bytes, the \C escape sequence must be used if a true wildcard is desired. The . character will not match the new line character \n, which may be present in a binary value.

Parameters

- **row_key_regex_filter** (*bytes*) – A regular expression (RE2) to match cells from rows with row keys that satisfy this regex. For a CheckAndMutateRowRequest, this filter is unnecessary since the row key is already specified.
- **family_name_regex_filter** (*str*) – A regular expression (RE2) to match cells from columns in a given column family. For technical reasons, the regex must not contain the ':' character, even if it isnot being uses as a literal.
- **column_qualifier_regex_filter** (*bytes*) – A regular expression (RE2) to match cells from column that match this regex (irrespective of column family).
- **value_regex_filter** (*bytes*) – A regular expression (RE2) to match cells with values that match this regex.
- **column_range_filter** (*ColumnRange*) – Range of columns to limit cells to.
- **timestamp_range_filter** (*TimestampRange*) – Range of time that cells should match against.
- **value_range_filter** (*CellValueRange*) – Range of cell values to filter for.
- **cells_per_row_offset_filter** (*int*) – Skips the first N cells of the row.
- **cells_per_row_limit_filter** (*int*) – Matches only the first N cells of the row.
- **cells_per_column_limit_filter** (*int*) – Matches only the most recent N cells within each column. This filters a (family name, column) pair, based on timestamps of each cell.
- **row_sample_filter** (*float*) – Non-deterministic filter. Matches all cells from a row with probability p, and matches no cells from the row with probability 1-p. (Here, the probability p is row_sample_filter.)

- **strip_value_transformer** (*bool*) – If `True`, replaces each cell's value with the empty string. As the name indicates, this is more useful as a transformer than a generic query / filter.

Raises `TypeError` if not exactly one value set in the constructor.

to_pb()
Converts the *RowFilter* to a protobuf.

Return type `data_pb2.RowFilter`

Returns The converted current object.

class `gcloud_bigtable.row.`**RowFilterChain**(*filters=None*)
Bases: `object`

Chain of row filters.

Sends rows through several filters in sequence. The filters are "chained" together to process a row. After the first filter is applied, the second is applied to the filtered output and so on for subsequent filters.

Parameters **filters** (*list*) – List of *RowFilter*, *RowFilterChain*, *RowFilterUnion* and/or *ConditionalRowFilter*

to_pb()
Converts the *RowFilterChain* to a protobuf.

Return type `data_pb2.RowFilter`

Returns The converted current object.

class `gcloud_bigtable.row.`**RowFilterUnion**(*filters=None*)
Bases: `object`

Union of row filters.

Sends rows through several filters simultaneously, then merges / interleaves all the filtered results together.

If multiple cells are produced with the same column and timestamp, they will all appear in the output row in an unspecified mutual order.

Parameters **filters** (*list*) – List of *RowFilter*, *RowFilterChain*, *RowFilterUnion* and/or *ConditionalRowFilter*

to_pb()
Converts the *RowFilterUnion* to a protobuf.

Return type `data_pb2.RowFilter`

Returns The converted current object.

class `gcloud_bigtable.row.`**TimestampRange**(*start=None*, *end=None*)
Bases: `object`

Range of time with inclusive lower and exclusive upper bounds.

Parameters

- **start** (`datetime.datetime`) – (Optional) The (inclusive) lower bound of the timestamp range. If omitted, defaults to Unix epoch.
- **end** (`datetime.datetime`) – (Optional) The (exclusive) upper bound of the timestamp range. If omitted, defaults to "infinity" (no upper bound).

to_pb()
Converts the *TimestampRange* to a protobuf.

Return type `data_pb2.TimestampRange`

Returns The converted current object.

CHAPTER 15

Row Data

Container for Google Cloud Bigtable Cells and Streaming Row Contents.

class gcloud_bigtable.row_data.**Cell**(*value*, *timestamp*)
Bases: object

Representation of a Google Cloud Bigtable Cell.

Parameters

- **value** (*bytes*) – The value stored in the cell.
- **timestamp** (datetime.datetime) – The timestamp when the cell was stored.

classmethod from_pb(*cell_pb*)
Create a new cell from a Cell protobuf.

Parameters **cell_pb** (_generated.bigtable_data_pb2.Cell) – The protobuf to convert.

Return type *Cell*

Returns The cell corresponding to the protobuf.

class gcloud_bigtable.row_data.**PartialRowData**(*row_key*)
Bases: object

Representation of partial row in a Google Cloud Bigtable Table.

These are expected to be updated directly from a _generated.bigtable_service_messages_pb2.ReadRowsRespo

Parameters **row_key** (*bytes*) – The key for the row holding the (partial) data.

cells
Property returning all the cells accumulated on this partial row.

Return type dict

Returns Dictionary of the *Cell* objects accumulated. This dictionary has two-levels of keys (first for column families and second for column names/qualifiers within a family). For a given column, a list of *Cell* objects is stored.

clear()
Clears all cells that have been added.

committed
Getter for the committed status of the (partial) row.

Return type bool

Returns The committed status of the (partial) row.

row_key
Getter for the current (partial) row's key.

Return type bytes

Returns The current (partial) row's key.

update_from_read_rows(*read_rows_response_pb*)
Updates the current row from a `ReadRows` response.

Parameters **read_rows_response_pb** (`_generated.bigtable_service_messages_pb2.ReadRows` – A response streamed back as part of a `ReadRows` request.

Raises `ValueError` if the current partial row has already been committed, if the row key on the response doesn't match the current one or if there is a chunk encountered with an unexpected `ONEOF` protobuf property.

class `gcloud_bigtable.row_data.`**PartialRowsData**(*response_iterator*)
Bases: `object`

Convenience wrapper for consuming a `ReadRows` streaming response.

Parameters **response_iterator** (`grpc.framework.alpha._reexport._CancellableIterator`) – A streaming iterator returned from a `ReadRows` request.

cancel()
Cancels the iterator, closing the stream.

consume_all(*max_loops=None*)
Consume the streamed responses until there are no more.

This simply calls *`consume_next()`* until there are no more to consume.

Parameters **max_loops** (*int*) – (Optional) Maximum number of times to try to consume an additional `ReadRowsResponse`. You can use this to avoid long wait times.

consume_next()
Consumes the next `ReadRowsResponse` from the stream.

Parses the response and stores it as a *`PartialRowData`* in a dictionary owned by this object.

Raises `StopIteration` if the response iterator has no more responses to stream.

rows
Property returning all rows accumulated from the stream.

Return type dict

Returns Dictionary of *`PartialRowData`*.

CHAPTER 16

Google Cloud Bigtable: Python

This library is an alpha implementation of Google Cloud Bigtable and is closely related to gcloud-python.

API requests are sent to the Google Cloud Bigtable API via RPC over HTTP/2. In order to support this, we'll rely on gRPC. We are working with the gRPC team to rapidly make the install story more user-friendly.

Get started by learning about the *Client* on the Base for Everything page. If you have install questions, check out the project's README.

In the hierarchy of API concepts

- a *Client* owns a *Cluster*
- a *Cluster* owns a *Table*
- a *Table* owns a *ColumnFamily*
- a *Table* owns a *Row* (and all the cells in the row)

16.1 Indices and tables

- genindex
- modindex

g

gcloud_bigtable.client, 15
gcloud_bigtable.cluster, 25
gcloud_bigtable.column_family, 37
gcloud_bigtable.row, 47
gcloud_bigtable.row_data, 57
gcloud_bigtable.table, 33

A

ADMIN_SCOPE (in module gcloud_bigtable.client), 15
ALL_COLUMNS (gcloud_bigtable.row.Row attribute), 49
append_cell_value() (gcloud_bigtable.row.Row method), 49

C

cancel() (gcloud_bigtable.row_data.PartialRowsData method), 58
Cell (class in gcloud_bigtable.row_data), 57
cells (gcloud_bigtable.row_data.PartialRowData attribute), 57
CellValueRange (class in gcloud_bigtable.row), 47
clear() (gcloud_bigtable.row_data.PartialRowData method), 57
clear_modification_rules() (gcloud_bigtable.row.Row method), 49
clear_mutations() (gcloud_bigtable.row.Row method), 49
Client (class in gcloud_bigtable.client), 15
client (gcloud_bigtable.cluster.Cluster attribute), 25
client (gcloud_bigtable.column_family.ColumnFamily attribute), 38
client (gcloud_bigtable.row.Row attribute), 49
client (gcloud_bigtable.table.Table attribute), 33
Cluster (class in gcloud_bigtable.cluster), 25
cluster (gcloud_bigtable.table.Table attribute), 33
cluster() (gcloud_bigtable.client.Client method), 16
CLUSTER_ADMIN_HOST (in module gcloud_bigtable.client), 15
CLUSTER_ADMIN_PORT (in module gcloud_bigtable.client), 15
cluster_stub (gcloud_bigtable.client.Client attribute), 16
column_family() (gcloud_bigtable.table.Table method), 33
ColumnFamily (class in gcloud_bigtable.column_family), 37
ColumnRange (class in gcloud_bigtable.row), 47
commit() (gcloud_bigtable.row.Row method), 49
commit_modifications() (gcloud_bigtable.row.Row method), 50
committed (gcloud_bigtable.row_data.PartialRowData attribute), 57
ConditionalRowFilter (class in gcloud_bigtable.row), 48
consume_all() (gcloud_bigtable.row_data.PartialRowsData method), 58
consume_next() (gcloud_bigtable.row_data.PartialRowsData method), 58
create() (gcloud_bigtable.cluster.Cluster method), 25
create() (gcloud_bigtable.column_family.ColumnFamily method), 38
create() (gcloud_bigtable.table.Table method), 34
credentials (gcloud_bigtable.client.Client attribute), 16

D

DATA_API_HOST (in module gcloud_bigtable.client), 18
DATA_API_PORT (in module gcloud_bigtable.client), 18
DATA_SCOPE (in module gcloud_bigtable.client), 18
data_stub (gcloud_bigtable.client.Client attribute), 16
DEFAULT_TIMEOUT_SECONDS (in module gcloud_bigtable.client), 18
DEFAULT_USER_AGENT (in module gcloud_bigtable.client), 18
delete() (gcloud_bigtable.cluster.Cluster method), 26
delete() (gcloud_bigtable.column_family.ColumnFamily method), 38
delete() (gcloud_bigtable.row.Row method), 50
delete() (gcloud_bigtable.table.Table method), 34
delete_cell() (gcloud_bigtable.row.Row method), 51
delete_cells() (gcloud_bigtable.row.Row method), 51

F

filter (gcloud_bigtable.row.Row attribute), 51
from_pb() (gcloud_bigtable.cluster.Cluster class method), 26
from_pb() (gcloud_bigtable.row_data.Cell class method), 57

from_service_account_json() (gcloud_bigtable.client.Client class method), 16
from_service_account_p12() (gcloud_bigtable.client.Client class method), 17

G

GarbageCollectionRule (class in gcloud_bigtable.column_family), 39
GarbageCollectionRuleIntersection (class in gcloud_bigtable.column_family), 39
GarbageCollectionRuleUnion (class in gcloud_bigtable.column_family), 39
gcloud_bigtable.client (module), 15
gcloud_bigtable.cluster (module), 25
gcloud_bigtable.column_family (module), 37
gcloud_bigtable.row (module), 47
gcloud_bigtable.row_data (module), 57
gcloud_bigtable.table (module), 33

I

increment_cell_value() (gcloud_bigtable.row.Row method), 51

L

list_clusters() (gcloud_bigtable.client.Client method), 17
list_column_families() (gcloud_bigtable.table.Table method), 34
list_tables() (gcloud_bigtable.cluster.Cluster method), 26
list_zones() (gcloud_bigtable.client.Client method), 17

N

name (gcloud_bigtable.cluster.Cluster attribute), 26
name (gcloud_bigtable.column_family.ColumnFamily attribute), 38
name (gcloud_bigtable.table.Table attribute), 34

O

operation_finished() (gcloud_bigtable.cluster.Cluster method), 26
operations_stub (gcloud_bigtable.client.Client attribute), 17

P

PartialRowData (class in gcloud_bigtable.row_data), 57
PartialRowsData (class in gcloud_bigtable.row_data), 58
PROJECT_ENV_VAR (in module gcloud_bigtable.client), 18
project_id (gcloud_bigtable.client.Client attribute), 18
project_id (gcloud_bigtable.cluster.Cluster attribute), 27
project_name (gcloud_bigtable.client.Client attribute), 18

R

READ_ONLY_SCOPE (in module gcloud_bigtable.client), 18
read_row() (gcloud_bigtable.table.Table method), 35
read_rows() (gcloud_bigtable.table.Table method), 35
reload() (gcloud_bigtable.cluster.Cluster method), 27
rename() (gcloud_bigtable.table.Table method), 35
Row (class in gcloud_bigtable.row), 48
row() (gcloud_bigtable.table.Table method), 36
row_key (gcloud_bigtable.row.Row attribute), 52
row_key (gcloud_bigtable.row_data.PartialRowData attribute), 58
RowFilter (class in gcloud_bigtable.row), 52
RowFilterChain (class in gcloud_bigtable.row), 54
RowFilterUnion (class in gcloud_bigtable.row), 54
rows (gcloud_bigtable.row_data.PartialRowsData attribute), 58

S

sample_row_keys() (gcloud_bigtable.table.Table method), 36
set_cell() (gcloud_bigtable.row.Row method), 52
start() (gcloud_bigtable.client.Client method), 18
stop() (gcloud_bigtable.client.Client method), 18

T

Table (class in gcloud_bigtable.table), 33
table (gcloud_bigtable.column_family.ColumnFamily attribute), 38
table (gcloud_bigtable.row.Row attribute), 52
table() (gcloud_bigtable.cluster.Cluster method), 27
TABLE_ADMIN_HOST (in module gcloud_bigtable.client), 18
TABLE_ADMIN_PORT (in module gcloud_bigtable.client), 18
table_stub (gcloud_bigtable.client.Client attribute), 18
timeout_seconds (gcloud_bigtable.cluster.Cluster attribute), 27
timeout_seconds (gcloud_bigtable.column_family.ColumnFamily attribute), 38
timeout_seconds (gcloud_bigtable.row.Row attribute), 52
timeout_seconds (gcloud_bigtable.table.Table attribute), 36
TimestampRange (class in gcloud_bigtable.row), 54
to_pb() (gcloud_bigtable.column_family.GarbageCollectionRule method), 39
to_pb() (gcloud_bigtable.column_family.GarbageCollectionRuleIntersection method), 39
to_pb() (gcloud_bigtable.column_family.GarbageCollectionRuleUnion method), 39
to_pb() (gcloud_bigtable.row.CellValueRange method), 47
to_pb() (gcloud_bigtable.row.ColumnRange method), 48

to_pb() (gcloud_bigtable.row.ConditionalRowFilter method), 48
to_pb() (gcloud_bigtable.row.RowFilter method), 54
to_pb() (gcloud_bigtable.row.RowFilterChain method), 54
to_pb() (gcloud_bigtable.row.RowFilterUnion method), 54
to_pb() (gcloud_bigtable.row.TimestampRange method), 54

U

undelete() (gcloud_bigtable.cluster.Cluster method), 27
update() (gcloud_bigtable.cluster.Cluster method), 27
update() (gcloud_bigtable.column_family.ColumnFamily method), 38
update_from_read_rows() (gcloud_bigtable.row_data.PartialRowData method), 58