
Riak Python Client

Release 2.4.2

February 21, 2016

1	Tutorial	1
2	Installation	3
3	Development	5
4	Indices and tables	7
5	Contents	9
	Python Module Index	73

Tutorial

The tutorial documentation has been converted to the [Basho Docs](#) as the [Taste of Riak: Python](#). The old [tutorial](#) that used to live here has been moved to the [Github Wiki](#) and is likely out-of-date.

Installation

1. Ensure Riak installed & running. (`riak ping`)
2. Install the Python client:
 1. If you use [Pip](#), `pip install riak`.
 2. If you use [easy_install](#), run `easy_install riak`.
 3. You can download the package off [PyPI](#), extract it and run `python setup.py install`.

Development

All development is done on [Github](#). Use [Issues](#) to report problems or submit contributions.

Indices and tables

- `genindex`
- `search`

Contents

5.1 Client & Connections

To connect to a Riak cluster, you must create a *RiakClient* object. The default configuration connects to a single Riak node on `localhost` with the default ports. The below instantiation statements are all equivalent:

```
from riak import RiakClient, RiakNode

RiakClient()
RiakClient(protocol='http', host='127.0.0.1', http_port=8098)
RiakClient(nodes=[{'host': '127.0.0.1', 'http_port': 8098}])
RiakClient(protocol='http', nodes=[RiakNode()])
```

Note: Connections are not established until you attempt to perform an operation. If the host or port are incorrect, you will not get an error raised immediately.

The client maintains a connection pool behind the scenes, one for each protocol. Connections are opened as-needed; a random node is selected when a new connection is requested.

5.1.1 Client objects

class `riak.client.RiakClient` (*protocol*='pbc', *transport_options*={}, *nodes*=None, *credentials*=None, *multiget_pool_size*=None, ***unused_args*)

The *RiakClient* object holds information necessary to connect to Riak. Requests can be made to Riak directly through the client or by using the methods on related objects.

Construct a new *RiakClient* object.

Parameters

- **protocol** (*string*) – the preferred protocol, defaults to ‘pbc’
- **nodes** (*list*) – a list of node configurations, where each configuration is a dict containing the keys ‘host’, ‘http_port’, and ‘pb_port’
- **transport_options** (*dict*) – Optional key-value args to pass to the transport constructor
- **credentials** (*SecurityCreds* or dict) – optional object of security info
- **multiget_pool_size** (*int*) – the number of threads to use in *multiget()* operations. Defaults to a factor of the number of CPUs in the system

PROTOCOLS = ['http', 'pb']

The supported protocols

Prior to Riak 2.0 the 'https' protocol was also an option, but now secure connections are handled by the *Security* feature.

protocol

Which protocol to prefer, one of *PROTOCOLS*. Please note that when one protocol is selected, the other protocols MAY NOT attempt to connect. Changing to another protocol will cause a connection on the next request.

Some requests are only valid over 'http', and will always be sent via those transports, regardless of which protocol is preferred.

client_id

The client ID for this client instance

resolver

The sibling-resolution function for this client. Defaults to *riak.resolver.default_resolver()*.

nodes

The list of *nodes* that this client will connect to. It is best not to modify this property directly, as it is not thread-safe.

Nodes

The *nodes* attribute of *RiakClient* objects is a list of *RiakNode* objects. If you include multiple host specifications in the *RiakClient* constructor, they will be turned into this type.

class *riak.node.RiakNode* (*host*='127.0.0.1', *http_port*=8098, *pb_port*=8087, ***unused_args*)

The internal representation of a Riak node to which the client can connect. Encapsulates both the configuration for the node and error tracking used for node-selection.

Creates a node.

Parameters

- **host** (*string*) – an IP address or hostname
- **http_port** (*integer*) – the HTTP port of the node
- **pb_port** (*integer*) – the Protocol Buffers port of the node

Retry logic

Some operations that fail because of network errors or Riak node failure may be safely retried on another node, and the client will do so automatically. The items below can be used to configure this behavior.

RiakClient.**retries**

The number of times retryable operations will be attempted before raising an exception to the caller. Defaults to 3.

Note This is a thread-local for safety and operation-specific modification. To change the default globally, modify *riak.client.transport.DEFAULT_RETRY_COUNT*.

RiakClient.**retry_count** (*retries*)

Modifies the number of retries for the scope of the *with* statement (in the current thread).

Example:

```
with client.retry_count(10):
    client.ping()
```

`riak.client.transport.DEFAULT_RETRY_COUNT = 3`

The default (global) number of times to retry requests that are retryable. This can be modified locally, per-thread, via the `RiakClient.retries` property, or using the `RiakClient.retry_count` method in a `with` statement.

5.1.2 Client-level Operations

Some operations are not scoped by buckets or bucket types and can be performed on the client directly:

`RiakClient.ping()`

Check if the Riak server for this `RiakClient` instance is alive.

Note: This request is automatically retried `retries` times if it fails due to network error.

Return type `boolean`

`RiakClient.get_buckets(bucket_type=None, timeout=None)`

Get the list of buckets as `RiakBucket` instances.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

Note: This request is automatically retried `retries` times if it fails due to network error.

Parameters

- **bucket_type** (`BucketType`) – the optional containing bucket type
- **timeout** (`int`) – a timeout value in milliseconds

Return type list of `RiakBucket` instances

`RiakClient.stream_buckets(bucket_type=None, timeout=None)`

Streams the list of buckets. This is a generator method that should be iterated over.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

The caller should explicitly close the returned iterator, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire iterator will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
with closing(client.stream_buckets()) as buckets:
    for bucket_list in buckets:
        do_something(bucket_list)

# Explicit close()
stream = client.stream_buckets()
```

```
for bucket_list in stream:
    do_something(bucket_list)
stream.close()
```

Parameters

- **bucket_type** (*BucketType*) – the optional containing bucket type
- **timeout** (*int*) – a timeout value in milliseconds

Return type iterator that yields lists of *RiakBucket* instances

5.1.3 Accessing Bucket Types and Buckets

Most client operations are on *bucket type objects*, the *bucket objects* they contain or keys within those buckets. Use the `bucket_type` or `bucket` methods for creating bucket types and buckets that will proxy operations to the called client.

`RiakClient.bucket_type(name)`

Gets the bucket-type by the specified name. Bucket-types do not always exist (unlike buckets), but this will always return a *BucketType* object.

Parameters **name** (*str*) – the bucket-type name

Return type *BucketType*

`RiakClient.bucket(name, bucket_type='default')`

Get the bucket by the specified name. Since buckets always exist, this will always return a *RiakBucket*.

If you are using a bucket that is contained in a bucket type, it is preferable to access it from the bucket type object:

```
# Preferred:
client.bucket_type("foo").bucket("bar")

# Equivalent, but not preferred:
client.bucket("bar", bucket_type="foo")
```

Parameters

- **name** (*str*) – the bucket name
- **bucket_type** (*BucketType* or *str*) – the parent bucket-type

Return type *RiakBucket*

5.1.4 Bucket Type Operations

`RiakClient.get_bucket_type_props(bucket_type)`

Fetches properties for the given bucket-type.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters **bucket_type** (*BucketType*) – the bucket-type whose properties will be fetched

Return type dict

`RiakClient.set_bucket_type_props(bucket_type, props)`
Sets properties for the given bucket-type.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket_type** (*BucketType*) – the bucket-type whose properties will be set
- **props** (*dict*) – the properties to set

5.1.5 Bucket Operations

`RiakClient.get_bucket_props(bucket)`
Fetches bucket properties for the given bucket.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters **bucket** (*RiakBucket*) – the bucket whose properties will be fetched

Return type *dict*

`RiakClient.set_bucket_props(bucket, props)`
Sets bucket properties for the given bucket.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket** (*RiakBucket*) – the bucket whose properties will be set
- **props** (*dict*) – the properties to set

`RiakClient.clear_bucket_props(bucket)`
Resets bucket properties for the given bucket.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters **bucket** (*RiakBucket*) – the bucket whose properties will be set

`RiakClient.get_keys(bucket, timeout=None)`
Lists all keys in a bucket.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket** (*RiakBucket*) – the bucket whose keys are fetched
- **timeout** (*int*) – a timeout value in milliseconds

Return type list

`RiakClient.stream_keys(bucket, timeout=None)`

Lists all keys in a bucket via a stream. This is a generator method which should be iterated over.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

The caller should explicitly close the returned iterator, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire iterator will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
with closing(client.stream_keys(mybucket)) as keys:
    for key_list in keys:
        do_something(key_list)

# Explicit close()
stream = client.stream_keys(mybucket)
for key_list in stream:
    do_something(key_list)
stream.close()
```

Parameters

- **bucket** (*RiakBucket*) – the bucket whose properties will be set
- **timeout** (*int*) – a timeout value in milliseconds

Return type iterator

5.1.6 Key-level Operations

`RiakClient.get(robj, r=None, pr=None, timeout=None)`

Fetches the contents of a Riak object.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **robj** (*RiakObject*) – the object to fetch
- **r** (*integer, string, None*) – the read quorum
- **pr** (*integer, string, None*) – the primary read quorum
- **timeout** (*int*) – a timeout value in milliseconds
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful

`RiakClient.put(robj, w=None, dw=None, pw=None, return_body=None, if_none_match=None, timeout=None)`
Stores an object in the Riak cluster.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **robj** (*RiakObject*) – the object to store
- **w** (*integer, string, None*) – the write quorum
- **dw** (*integer, string, None*) – the durable write quorum
- **pw** (*integer, string, None*) – the primary write quorum
- **return_body** (*boolean*) – whether to return the resulting object after the write
- **if_none_match** (*boolean*) – whether to fail the write if the object exists
- **timeout** (*int*) – a timeout value in milliseconds

`RiakClient.delete(robj, rw=None, r=None, w=None, dw=None, pr=None, pw=None, timeout=None)`
Deletes an object from Riak.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **robj** (*RiakObject*) – the object to delete
- **rw** (*integer, string, None*) – the read/write (delete) quorum
- **r** (*integer, string, None*) – the read quorum
- **pr** (*integer, string, None*) – the primary read quorum
- **w** (*integer, string, None*) – the write quorum
- **dw** (*integer, string, None*) – the durable write quorum
- **pw** (*integer, string, None*) – the primary write quorum
- **timeout** (*int*) – a timeout value in milliseconds

`RiakClient.multiget(pairs, **params)`
Fetches many keys in parallel via threads.

Parameters

- **pairs** (*list*) – list of bucket_type/bucket/key tuple triples
- **params** (*dict*) – additional request flags, e.g. r, pr

Return type list of *RiakObjects*, *Datatypes*, or tuples of bucket_type, bucket, key, and the exception raised on fetch

`RiakClient.fetch_datatype(bucket, key, r=None, pr=None, basic_quorum=None, not_found_ok=None, timeout=None, include_context=None)`
Fetches the value of a Riak Datatype.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket** (*RiakBucket*) – the bucket of the datatype, which must belong to a *BucketType*
- **key** (*string*) – the key of the datatype
- **r** (*integer, string, None*) – the read quorum
- **pr** (*integer, string, None*) – the primary read quorum
- **basic_quorum** (*bool, None*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool, None*) – whether to treat not-found responses as successful
- **timeout** (*int, None*) – a timeout value in milliseconds
- **include_context** (*bool, None*) – whether to return the opaque context as well as the value, which is useful for removal operations on sets and maps

Return type *Datatype*

`RiakClient.update_datatype(datatype, w=None, dw=None, pw=None, return_body=None, timeout=None, include_context=None)`

Sends an update to a Riak Datatype to the server. This operation is not idempotent and so will not be retried automatically.

Parameters

- **datatype** (*Datatype*) – the datatype with pending updates
- **w** (*integer, string, None*) – the write quorum
- **dw** (*integer, string, None*) – the durable write quorum
- **pw** (*integer, string, None*) – the primary write quorum
- **timeout** (*int*) – a timeout value in milliseconds
- **include_context** (*bool*) – whether to return the opaque context as well as the value, which is useful for removal operations on sets and maps

Return type tuple of datatype, opaque value and opaque context

5.1.7 Timeseries Operations

`RiakClient.ts_describe(table)`

Retrieve a time series table description from the Riak cluster.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters **table** (string or Table) – The timeseries table.

Return type TsObject

`RiakClient.ts_get(table, key)`
Retrieve timeseries value by key

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **table** (string or Table) – The timeseries table.
- **key** (*list*) – The timeseries value’s key.

Return type TsObject

`RiakClient.ts_put(tsobj)`
Stores time series data in the Riak cluster.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters **tsobj** (*RiakTsObject*) – the time series object to store

Return type boolean

`RiakClient.ts_delete(table, key)`
Delete timeseries value by key

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **table** (string or Table) – The timeseries table.
- **key** (*list or dict*) – The timeseries value’s key.

Return type boolean

`RiakClient.ts_query(table, query, interpolations=None)`
Queries time series data in the Riak cluster.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **table** (string or Table) – The timeseries table.
- **query** (*string*) – The timeseries query.

Return type TsObject

`RiakClient.ts_stream_keys(table, timeout=None)`
Lists all keys in a time series table via a stream. This is a generator method which should be iterated over.

The caller should explicitly close the returned iterator, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire iterator will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
with closing(client.ts_stream_keys(mytable)) as keys:
    for key_list in keys:
        do_something(key_list)

# Explicit close()
stream = client.ts_stream_keys(mytable)
for key_list in stream:
    do_something(key_list)
stream.close()
```

Parameters

- **table** (*Table*) – the table from which to stream keys
- **timeout** (*int*) – a timeout value in milliseconds

Return type iterator

5.1.8 Query Operations

`RiakClient.mapred(inputs, query, timeout)`
 Executes a MapReduce query.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **inputs** (*list, dict*) – the input list/structure
- **query** (*list*) – the list of query phases
- **timeout** (*integer, None*) – the query timeout

Return type *mixed*

`RiakClient.stream_mapred(inputs, query, timeout)`
 Streams a MapReduce query as (phase, data) pairs. This is a generator method which should be iterated over.

The caller should explicitly close the returned iterator, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire iterator will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
with closing(mymapred.stream()) as results:
    for phase, result in results:
        do_something(phase, result)

# Explicit close()
```

```
stream = mymapred.stream()
for phase, result in stream:
    do_something(phase, result)
stream.close()
```

Parameters

- **inputs** (*list*, *dict*) – the input list/structure
- **query** (*list*) – the list of query phases
- **timeout** (*integer*, *None*) – the query timeout

Return type *iterator*

`RiakClient.get_index(bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None)`
 Queries a secondary index, returning matching keys.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket** (*RiakBucket*) – the bucket whose index will be queried
- **index** (*string*) – the index to query
- **startkey** (*string*, *integer*) – the sole key to query, or beginning of the query range
- **endkey** (*string*, *integer*) – the end of the query range (optional if equality)
- **return_terms** (*boolean*) – whether to include the secondary index value
- **max_results** (*integer*) – the maximum number of results to return (page size)
- **continuation** (*string*) – the opaque continuation returned from a previous paginated request
- **timeout** (*int*) – a timeout value in milliseconds, or ‘infinity’
- **term_regex** (*string*) – a regular expression used to filter index terms

Return type *IndexPage*

`RiakClient.stream_index(bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None)`

Queries a secondary index, streaming matching keys through an iterator.

The caller should explicitly close the returned iterator, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire iterator will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
with closing(client.stream_index(mybucket, 'name_bin',
                                'Smith')) as index:
    for key in index:
        do_something(key)
```

```
# Explicit close()
stream = client.stream_index(mybucket, 'name_bin', 'Smith')
for key in stream:
    do_something(key)
stream.close()
```

Parameters

- **bucket** (*RiakBucket*) – the bucket whose index will be queried
- **index** (*string*) – the index to query
- **startkey** (*string, integer*) – the sole key to query, or beginning of the query range
- **endkey** (*string, integer*) – the end of the query range (optional if equality)
- **return_terms** (*boolean*) – whether to include the secondary index value
- **max_results** (*integer*) – the maximum number of results to return (page size)
- **continuation** (*string*) – the opaque continuation returned from a previous paginated request
- **timeout** (*int*) – a timeout value in milliseconds, or ‘infinity’
- **term_regex** (*string*) – a regular expression used to filter index terms

Return type *IndexPage*

`RiakClient.fulltext_search(index, query, **params)`

Performs a full-text search query.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **index** (*string*) – the bucket/index to search over
- **query** (*string*) – the search query
- **params** (*dict*) – additional query flags

Return type *dict*

`RiakClient.paginate_index(bucket, index, startkey, endkey=None, max_results=1000, return_terms=None, continuation=None, timeout=None, term_regex=None)`

Iterates over a paginated index query. This is equivalent to calling *get_index()* and then successively calling *next_page()* until all results are exhausted.

Because limiting the result set is necessary to invoke pagination, the `max_results` option has a default of 1000.

Parameters

- **bucket** (*RiakBucket*) – the bucket whose index will be queried
- **index** (*string*) – the index to query
- **startkey** (*string, integer*) – the sole key to query, or beginning of the query range
- **endkey** (*string, integer*) – the end of the query range (optional if equality)

- **return_terms** (*boolean*) – whether to include the secondary index value
- **max_results** (*integer*) – the maximum number of results to return (page size), defaults to 1000
- **continuation** (*string*) – the opaque continuation returned from a previous paginated request
- **timeout** (*int*) – a timeout value in milliseconds, or ‘infinity’
- **term_regex** (*string*) – a regular expression used to filter index terms

Return type generator over instances of *IndexPage*

`RiakClient.paginate_stream_index(bucket, index, startkey, endkey=None, max_results=1000, return_terms=None, continuation=None, timeout=None, term_regex=None)`

Iterates over a streaming paginated index query. This is equivalent to calling `stream_index()` and then successively calling `next_page()` until all results are exhausted.

Because limiting the result set is necessary to invoke pagination, the `max_results` option has a default of 1000.

The caller should explicitly close each yielded page, either using `contextlib.closing()` or calling `close()` explicitly. Consuming the entire page will also close the stream. If it does not, the associated connection might not be returned to the pool. Example:

```
from contextlib import closing

# Using contextlib.closing
for page in client.paginate_stream_index(mybucket, 'name_bin',
                                         'Smith'):
    with closing(page):
        for key in page:
            do_something(key)

# Explicit close()
for page in client.paginate_stream_index(mybucket, 'name_bin',
                                         'Smith'):
    for key in page:
        do_something(key)
    page.close()
```

Parameters

- **bucket** (*RiakBucket*) – the bucket whose index will be queried
- **index** (*string*) – the index to query
- **startkey** (*string, integer*) – the sole key to query, or beginning of the query range
- **endkey** (*string, integer*) – the end of the query range (optional if equality)
- **return_terms** (*boolean*) – whether to include the secondary index value
- **max_results** (*integer*) – the maximum number of results to return (page size), defaults to 1000
- **continuation** (*string*) – the opaque continuation returned from a previous paginated request
- **timeout** (*int*) – a timeout value in milliseconds, or ‘infinity’
- **term_regex** (*string*) – a regular expression used to filter index terms

Return type generator over instances of *IndexPage*

5.1.9 Search Maintenance Operations

`RiakClient.create_search_schema(schema, content)`

Creates a Solr schema of the given name and content. Content must be valid Solr schema XML.

Parameters

- **schema** (*string*) – the name of the schema to create
- **content** (*string*) – the solr schema xml content

`RiakClient.get_search_schema(schema)`

Gets a search schema of the given name if it exists. Raises a `RiakError` if no such schema exists. The schema is returned as a dict with keys 'name' and 'content'.

Parameters **schema** (*string*) – the name of the schema to get

Returns dict

`RiakClient.create_search_index(index, schema=None, n_val=None)`

Create a search index of the given name, and optionally set a schema. If no schema is set, the default will be used.

Parameters

- **index** (*string*) – the name of the index to create
- **schema** (*string, None*) – the schema that this index will follow
- **n_val** (*integer, None*) – this indexes N value
- **timeout** (*integer, None*) – optional timeout (in ms)

`RiakClient.get_search_index(index)`

Gets a search index of the given name if it exists, which will also return the schema. Raises a `RiakError` if no such schema exists. The returned dict contains keys 'name', 'schema' and 'n_val'.

Parameters **index** (*string*) – the name of the index to create

Return type dict

`RiakClient.delete_search_index(index)`

Delete the search index that matches the given name.

Parameters **index** (*string*) – the name of the index to delete

`RiakClient.list_search_indexes()`

Gets all search indexes and their schemas. The returned list contains dicts with keys 'name', 'schema' and 'n_val'.

Returns list of dicts

5.1.10 Serialization

The client supports automatic transformation of Riak responses into Python types if encoders and decoders are registered for the media-types. Supported by default are `application/json` and `text/plain`.

`riak.client.default_encoder(obj)`

Default encoder for JSON datatypes, which returns UTF-8 encoded json instead of the default bloated backslash u XXXX escaped ASCII strings.

`RiakClient.get_encoder(content_type)`

Get the encoding function for the provided content type.

Parameters `content_type` (*str*) – the requested media type

Return type function

`RiakClient.set_encoder(content_type, encoder)`

Set the encoding function for the provided content type.

Parameters

- **content_type** (*str*) – the requested media type
- **encoder** (*function*) – an encoding function, takes a single object argument and returns encoded data

`RiakClient.get_decoder(content_type)`

Get the decoding function for the provided content type.

Parameters `content_type` (*str*) – the requested media type

Return type function

`RiakClient.set_decoder(content_type, decoder)`

Set the decoding function for the provided content type.

Parameters

- **content_type** (*str*) – the requested media type
- **decoder** (*function*) – a decoding function, takes encoded data and returns a Python type

5.1.11 Deprecated Features

Full-text search

The original version of Riak Search has been replaced by *Riak Search 2.0 (Yokozuna)*, which is full-blown Solr integration with Riak.

If Riak Search 1.0 is enabled, you can query an index via the bucket's `search()` method:

```
bucket.enable_search()
bucket.new("one", data={'value': 'one'},
          content_type="application/json").store()

bucket.search('value=one')
```

To manually add and remove documents from an index (without an associated key), use the *RiakClient* `fulltext_add()` and `fulltext_delete()` methods directly.

`RiakClient.fulltext_add(index, docs)`

Deprecated since version 2.1.0: (Riak 2.0) Manual index maintenance is not supported for *Riak Search 2.0*.

Adds documents to the full-text index.

Note: This request is automatically retried *retries* times if it fails due to network error. Only HTTP will be used for this request.

Parameters

- **index** (*string*) – the bucket/index in which to index these docs
- **docs** (*list*) – the list of documents

`RiakClient.fulltext_delete(index, docs=None, queries=None)`

Deprecated since version 2.1.0: (Riak 2.0) Manual index maintenance is not supported for [Riak Search 2.0](#).

Removes documents from the full-text index.

Note: This request is automatically retried *retries* times if it fails due to network error. Only HTTP will be used for this request.

Parameters

- **index** (*string*) – the bucket/index from which to delete
- **docs** (*list*) – a list of documents (with ids)
- **queries** (*list*) – a list of queries to match and delete

Legacy Counters

The first Data Type introduced in Riak 1.4 were *counters*. These pre-date *Bucket Types* and the current implementation. Rather than returning objects, the counter operations act directly on the value of the counter. Legacy counters are deprecated as of Riak 2.0. Please use *Counter* instead.

Warning: Legacy counters are incompatible with Bucket Types.

`RiakClient.get_counter(bucket, key, r=None, pr=None, basic_quorum=None, notfound_ok=None)`

Gets the value of a counter.

Deprecated since version 2.1.0: (Riak 2.0) Riak 1.4-style counters are deprecated in favor of the *Counter* datatype.

Note: This request is automatically retried *retries* times if it fails due to network error.

Parameters

- **bucket** (*RiakBucket*) – the bucket of the counter
- **key** (*string*) – the key of the counter
- **r** (*integer, string, None*) – the read quorum
- **pr** (*integer, string, None*) – the primary read quorum
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful

Return type integer

`RiakClient.update_counter(bucket, key, value, w=None, dw=None, pw=None, returnvalue=False)`

Deprecated since version 2.1.0: (Riak 2.0) Riak 1.4-style counters are deprecated in favor of the [Counter](#) datatype.

Updates a counter by the given value. This operation is not idempotent and so should not be retried automatically.

Parameters

- **bucket** ([RiakBucket](#)) – the bucket of the counter
- **key** (*string*) – the key of the counter
- **value** (*integer*) – the amount to increment or decrement
- **w** (*integer, string, None*) – the write quorum
- **dw** (*integer, string, None*) – the durable write quorum
- **pw** (*integer, string, None*) – the primary write quorum
- **returnvalue** (*bool*) – whether to return the updated value of the counter

5.2 Buckets & Bucket Types

Buckets are both namespaces for the key-value pairs you store in Riak, and containers for properties that apply to that namespace. In older versions of Riak, this was the only logical organization available. Now a higher-level collection called a **Bucket Type** can group buckets together. They allow for efficiently setting properties on a group of buckets at the same time.

Unlike buckets, Bucket Types must be [explicitly created](#) and activated before being used:

```
riak-admin bucket-type create n_equals_1 '{"props":{"n_val":1}}'
riak-admin bucket-type activate n_equals_1
```

Bucket Type creation and activation is only supported via the `riak-admin bucket-type` command-line tool. Riak 2.0 does not include an API to perform these actions, but the Python client *can* [retrieve](#) and [set](#) bucket-type properties.

If Bucket Types are not specified, the *default* bucket type is used. These buckets should be created via the [bucket\(\)](#) method on the client object, like so:

```
import riak

client = riak.RiakClient()
mybucket = client.bucket('mybucket')
```

Buckets with a user-specified Bucket Type can also be created via the same [bucket\(\)](#) method with an additional parameter or explicitly via [bucket_type\(\)](#):

```
othertype = client.bucket_type('othertype')
otherbucket = othertype.bucket('otherbucket')

# Alternate way to get a bucket within a bucket-type
mybucket = client.bucket('mybucket', bucket_type='mybuckettype')
```

For more detailed discussion, see [Using Bucket Types](#).

5.2.1 Bucket objects

class `riak.bucket.RiakBucket` (*client*, *name*, *bucket_type*)

The `RiakBucket` object allows you to access and change information about a Riak bucket, and provides methods to create or retrieve objects within the bucket.

Returns a new `RiakBucket` instance.

Parameters

- **client** (*RiakClient*) – A *RiakClient* instance
- **name** (*string*) – The bucket name
- **bucket_type** (*BucketType*) – The parent bucket type of this bucket

name

The name of the bucket, a string.

bucket_type

The parent *BucketType* for the bucket.

resolver

The sibling-resolution function for this bucket. If the resolver is not set, the client's resolver will be used.

5.2.2 Bucket properties

Bucket properties are flags and defaults that apply to all keys in the bucket.

`RiakBucket.get_properties()`

Retrieve a dict of all bucket properties.

Return type dict

`RiakBucket.set_properties(props)`

Set multiple bucket properties in one call.

Parameters *props* (*dict*) – A dictionary of properties

`RiakBucket.clear_properties()`

Reset all bucket properties to their defaults.

`RiakBucket.get_property(key)`

Retrieve a bucket property.

Parameters *key* (*string*) – The property to retrieve.

Return type *mixed*

`RiakBucket.set_property(key, value)`

Set a bucket property.

Parameters

- **key** (*string*) – Property to set.
- **value** (*mixed*) – Property value.

Shortcuts for common properties

Some of the most commonly-used bucket properties are exposed as object properties as well. The getters and setters simply call `RiakBucket.get_property()` and `RiakBucket.set_property()` respectively.

`RiakBucket.n_val`

N-value for this bucket, which is the number of replicas that will be written of each object in the bucket.

Warning: Set this once before you write any data to the bucket, and never change it again, otherwise unpredictable things could happen. This should only be used if you know what you are doing.

`RiakBucket.allow_mult`

If set to True, then writes with conflicting data will be stored and returned to the client.

`RiakBucket.r`

The default 'read' quorum for this bucket (how many replicas must reply for a successful read). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

`RiakBucket.pr`

The default 'primary read' quorum for this bucket (how many primary replicas are required for a successful read). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

`RiakBucket.w`

The default 'write' quorum for this bucket (how many replicas must acknowledge receipt of a write). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

`RiakBucket.dw`

The default 'durable write' quorum for this bucket (how many replicas must commit the write). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

`RiakBucket.pw`

The default 'primary write' quorum for this bucket (how many primary replicas are required for a successful write). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

`RiakBucket.rw`

The default 'read' and 'write' quorum for this bucket (equivalent to 'r' and 'w' but for deletes). This should be an integer less than the 'n_val' property, or a string of 'one', 'quorum', 'all', or 'default'

5.2.3 Working with keys

The primary purpose of buckets is to act as namespaces for keys. As such, you can use the bucket object to create, fetch and delete *objects*.

`RiakBucket.new` (*key=None, data=None, content_type='application/json', encoded_data=None*)

A shortcut for manually instantiating a new *RiakObject* or a new *Datatype*, based on the presence and value of the *datatype* bucket property. When the bucket contains a *Datatype*, all arguments are ignored except *key*, otherwise they are used to initialize the *RiakObject*.

Parameters

- **key** (*str*) – Name of the key. Leaving this to be None (default) will make Riak generate the key on store.
- **data** (*object*) – The data to store in a *RiakObject*, see *RiakObject.data*.
- **content_type** (*str*) – The media type of the data stored in the *RiakObject*, see *RiakObject.content_type*.
- **encoded_data** (*str*) – The encoded data to store in a *RiakObject*, see *RiakObject.encoded_data*.

Return type *RiakObject* or *Datatype*

`RiakBucket.new_from_file(key, filename)`

Create a new Riak object in the bucket, using the contents of the specified file. This is a shortcut for `new()`, where the `encoded_data` and `content_type` are set for you.

Warning: This is not supported for buckets that contain *Datatypes*.

Parameters

- **key** (*string*) – the key of the new object
- **filename** (*string*) – the file to read the contents from

Return type *RiakObject*

`RiakBucket.get(key, r=None, pr=None, timeout=None, include_context=None, basic_quorum=None, notfound_ok=None)`

Retrieve a *RiakObject* or *Datatype*, based on the presence and value of the *datatype* bucket property.

Parameters

- **key** (*string*) – Name of the key.
- **r** (*integer*) – R-Value of the request (defaults to bucket’s R)
- **pr** (*integer*) – PR-Value of the request (defaults to bucket’s PR)
- **timeout** (*int*) – a timeout value in milliseconds
- **include_context** (*bool*) – if the bucket contains datatypes, include the opaque context in the result
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful

Return type *RiakObject* or *Datatype*

`RiakBucket.multiget(keys, r=None, pr=None, timeout=None, basic_quorum=None, notfound_ok=None)`

Retrieves a list of keys belonging to this bucket in parallel.

Parameters

- **keys** (*list*) – the keys to fetch
- **r** (*integer*) – R-Value for the requests (defaults to bucket’s R)
- **pr** (*integer*) – PR-Value for the requests (defaults to bucket’s PR)
- **timeout** (*int*) – a timeout value in milliseconds
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful

Return type list of *RiakObjects*, *Datatypes*, or tuples of `bucket_type`, `bucket`, `key`, and the exception raised on fetch

`RiakBucket.delete(key, **kwargs)`

Deletes a key from Riak. Short hand for `bucket.new(key).delete()`. See *RiakClient.delete()* for options.

Parameters **key** (*string*) – The key for the object

Return type *RiakObject*

5.2.4 Query operations

`RiakBucket.search(query, index=None, **params)`

Queries a search index over objects in this bucket/index. See `RiakClient.fulltext_search()` for more details.

Parameters

- **query** (*string*) – the search query
- **index** (*string or None*) – the index to search over. Defaults to the bucket's name.
- **params** (*dict*) – additional query flags

`RiakBucket.get_index(index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None)`

Queries a secondary index over objects in this bucket, returning keys or index/key pairs. See `RiakClient.get_index()` for more details.

`RiakBucket.stream_index(index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None)`

Queries a secondary index over objects in this bucket, streaming keys or index/key pairs via an iterator. The caller must close the stream when finished. See `RiakClient.stream_index()` for more details.

`RiakBucket.paginate_index(index, startkey, endkey=None, return_terms=None, max_results=1000, continuation=None, timeout=None, term_regex=None)`

Paginates through a secondary index over objects in this bucket, returning keys or index/key pairs. See `RiakClient.paginate_index()` for more details.

`RiakBucket.paginate_stream_index(index, startkey, endkey=None, return_terms=None, max_results=1000, continuation=None, timeout=None, term_regex=None)`

Paginates through a secondary index over objects in this bucket, streaming keys or index/key pairs. The caller must close the stream when finished. See `RiakClient.paginate_stream_index()` for more details.

5.2.5 Serialization

Similar to `RiakClient`, buckets can register custom transformation functions for media-types. When undefined on the bucket, `RiakBucket.get_encoder()` and `RiakBucket.get_decoder()` will delegate to the client associated with the bucket.

`RiakBucket.get_encoder(content_type)`

Get the encoding function for the provided content type for this bucket.

Parameters

- **content_type** (*str*) – the requested media type
- **content_type** – Content type requested

`RiakBucket.set_encoder(content_type, encoder)`

Set the encoding function for the provided content type for this bucket.

Parameters

- **content_type** (*str*) – the requested media type
- **encoder** (*function*) – an encoding function, takes a single object argument and returns a string data as single argument.

`RiakBucket.get_decoder(content_type)`

Get the decoding function for the provided content type for this bucket.

Parameters `content_type` (*str*) – the requested media type

Return type function

`RiakBucket.set_decoder(content_type, decoder)`

Set the decoding function for the provided content type for this bucket.

Parameters

- **content_type** (*str*) – the requested media type
- **decoder** (*function*) – a decoding function, takes a string and returns a Python type

5.2.6 Listing keys

Shortcuts for `RiakClient.get_keys()` and `RiakClient.stream_keys()` are exposed on the bucket object. The same admonitions for these operations apply.

`RiakBucket.get_keys()`

Return all keys within the bucket.

Return type list of keys

`RiakBucket.stream_keys()`

Streams all keys within the bucket through an iterator.

The caller must close the stream when finished. See `RiakClient.stream_keys()` for more details.

Return type iterator

5.2.7 Bucket Type objects

`class riak.bucket.BucketType(client, name)`

The `BucketType` object allows you to access and change properties on a Riak bucket type and access buckets within its namespace.

Returns a new `BucketType` instance.

Parameters

- **client** (*RiakClient*) – A *RiakClient* instance
- **name** (*string*) – The bucket-type's name

name

The name of the Bucket Type, a string.

`BucketType.is_default()`

Whether this bucket type is the default type, or a user-defined type.

Return type bool

`BucketType.bucket(name)`

Gets a bucket that belongs to this bucket-type.

Parameters **name** (*str*) – the bucket name

Return type *RiakBucket*

5.2.8 Bucket Type properties

Bucket Type properties are flags and defaults that apply to all buckets in the Bucket Type.

`BucketType.get_properties()`

Retrieve a dict of all bucket-type properties.

Return type dict

`BucketType.set_properties(props)`

Set multiple bucket-type properties in one call.

Parameters `props` (dict) – A dictionary of properties

`BucketType.get_property(key)`

Retrieve a bucket-type property.

Parameters `key` (string) – The property to retrieve.

Return type mixed

`BucketType.set_property(key, value)`

Set a bucket-type property.

Parameters

- **key** (string) – Property to set.
- **value** (mixed) – Property value.

`BucketType.datatype`

The assigned datatype for this bucket type, if present.

Return type None or str

5.2.9 Listing buckets

Shortcuts for `RiakClient.get_buckets()` and `RiakClient.stream_buckets()` are exposed on the bucket type object. This is similar to *Listing keys* on buckets.

`BucketType.get_buckets(timeout=None)`

Get the list of buckets under this bucket-type as `RiakBucket` instances.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

Note: This request is automatically retried `retries` times if it fails due to network error.

Parameters `timeout` (int) – a timeout value in milliseconds

Return type list of `RiakBucket` instances

`BucketType.stream_buckets(timeout=None)`

Streams the list of buckets under this bucket-type. This is a generator method that should be iterated over.

The caller must close the stream when finished. See `RiakClient.stream_buckets()` for more details.

Warning: Do not use this in production, as it requires traversing through all keys stored in a cluster.

Parameters `timeout` (int) – a timeout value in milliseconds

Return type iterator that yields lists of *RiakBucket* instances

5.2.10 Deprecated Features

Shortcuts for Riak Search 1.0

When Riak Search 1.0 is enabled on the server, you can toggle which buckets have automatic indexing turned on using the `search` bucket property (and on older versions, the `precommit` property). These methods simplify interacting with that configuration.

`RiakBucket.search_enabled()`

Returns True if search indexing is enabled for this bucket.

Deprecated since version 2.1.0: (Riak 2.0) Use *Riak Search 2.0* instead.

`RiakBucket.enable_search()`

Enable search indexing for this bucket.

Deprecated since version 2.1.0: (Riak 2.0) Use *Riak Search 2.0* instead.

`RiakBucket.disable_search()`

Disable search indexing for this bucket.

Deprecated since version 2.1.0: (Riak 2.0) Use *Riak Search 2.0* instead.

Legacy Counters

The `get_counter()` and `update_counter()`. See *Legacy Counters* for more details.

Warning: Legacy counters are incompatible with Bucket Types.

`RiakBucket.get_counter(key, **kwargs)`

Gets the value of a counter stored in this bucket. See *RiakClient.get_counter()* for options.

Deprecated since version 2.1.0: (Riak 2.0) Riak 1.4-style counters are deprecated in favor of the *Counter* datatype.

Parameters `key` (*string*) – the key of the counter

Return type `int`

`RiakBucket.update_counter(key, value, **kwargs)`

Updates the value of a counter stored in this bucket. Positive values increment the counter, negative values decrement. See *RiakClient.update_counter()* for options.

Deprecated since version 2.1.0: (Riak 2.0) Riak 1.4-style counters are deprecated in favor of the *Counter* datatype.

Parameters

- **key** (*string*) – the key of the counter
- **value** (*integer*) – the amount to increment or decrement

5.3 Values & Objects

Keys in Riak are namespaced into *buckets*, and their associated values are represented by *objects*, not to be confused with Python “objects”. A *RiakObject* is a container for the key, the *Vector clock*, the value(s) and any metadata associated with the value(s).

Values may also be *datatypes*, but are not discussed here.

5.3.1 RiakObject

class `riak.riak_object.RiakObject` (*client*, *bucket*, *key=None*)

The *RiakObject* holds meta information about a Riak object, plus the object’s data.

Construct a new *RiakObject*.

Parameters

- **client** (*RiakClient*) – A *RiakClient* object.
- **bucket** (*RiakBucket*) – A *RiakBucket* object.
- **key** (*string*) – An optional key. If not specified, then the key is generated by the server when *store()* is called.

key

The key of this object, a string. If not present, the server will generate a key the first time this object is stored.

bucket

The *bucket* to which this object belongs.

resolver

The sibling-resolution function for this object. If the resolver is not set, the bucket’s resolver will be used.

vclock

The *Vector clock* for this object.

exists

Whether the object exists. This is only `False` when there are no siblings (the object was not found), or the solitary sibling is a tombstone.

Vector clock

Vector clocks are Riak’s means of tracking the relationships between writes to a key. It is best practice to fetch the latest version of a key before attempting to modify or overwrite the value; if you do not, you may create *Siblings* or lose data! The content of a vector clock is essentially opaque to the user.

class `riak.riak_object.VClock` (*value*, *encoding*)

A representation of a vector clock received from Riak.

5.3.2 Persistence

Fetching, storing, and deleting keys are the bread-and-butter of Riak.

`RiakObject.store` (*w=None*, *dw=None*, *pw=None*, *return_body=True*, *if_none_match=False*, *time-out=None*)

Store the object in Riak. When this operation completes, the object could contain new metadata and possibly new data if Riak contains a newer version of the object according to the object’s vector clock.

Parameters

- **w** (*integer*) – W-value, wait for this many partitions to respond before returning to client.
- **dw** (*integer*) – DW-value, wait for this many partitions to confirm the write before returning to client.
- **pw** (*integer*) – PW-value, require this many primary partitions to be available before performing the put
- **return_body** (*bool*) – if the newly stored object should be retrieved
- **if_none_match** (*bool*) – Should the object be stored only if there is no key previously defined
- **timeout** (*int*) – a timeout value in milliseconds

Return type *RiakObject*

`RiakObject.reload(r=None, pr=None, timeout=None, basic_quorum=None, notfound_ok=None)`

Reload the object from Riak. When this operation completes, the object could contain new metadata and a new value, if the object was updated in Riak since it was last retrieved.

Note: Even if the key is not found in Riak, this will return a *RiakObject*. Check the *exists* property to see if the key was found.

Parameters

- **r** (*integer*) – R-Value, wait for this many partitions to respond before returning to client.
- **pr** (*integer*) – PR-value, require this many primary partitions to be available before performing the read that precedes the put
- **timeout** (*int*) – a timeout value in milliseconds
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful

Return type *RiakObject*

`RiakObject.delete(r=None, w=None, dw=None, pr=None, pw=None, timeout=None)`

Delete this object from Riak.

Parameters

- **r** (*integer*) – R-value, wait for this many partitions to read object before performing the put
- **w** (*integer*) – W-value, wait for this many partitions to respond before returning to client.
- **dw** (*integer*) – DW-value, wait for this many partitions to confirm the write before returning to client.
- **pr** (*integer*) – PR-value, require this many primary partitions to be available before performing the read that precedes the put
- **pw** (*integer*) – PW-value, require this many primary partitions to be available before performing the put
- **timeout** (*int*) – a timeout value in milliseconds

Return type *RiakObject*

5.3.3 Value and Metadata

Unless you have enabled *Siblings* via the *allow_mult* bucket property, you can inspect and manipulate the value and metadata of an object directly using these properties and methods:

`RiakObject.data`

The data stored in this object, as Python objects. For the raw data, use the *encoded_data* property. If unset, accessing this property will result in decoding the *encoded_data* property into Python values. The decoding is dependent on the *content_type* property and the bucket's registered decoders.

`RiakObject.encoded_data`

The raw data stored in this object, essentially the encoded form of the *data* property. If unset, accessing this property will result in encoding the *data* property into a string. The encoding is dependent on the *content_type* property and the bucket's registered encoders.

`RiakObject.content_type`

The MIME media type of the encoded data as a string

`RiakObject.charset`

The character set of the encoded data as a string

`RiakObject.content_encoding`

The encoding (compression) of the encoded data. Valid values are identity, deflate, gzip

`RiakObject.last_modified`

The UNIX timestamp of the modification time of this value.

`RiakObject.etag`

A unique entity-tag for the value.

`RiakObject.usermeta`

Arbitrary user-defined metadata dict, mapping strings to strings.

`RiakObject.links`

A set of bucket/key/tag 3-tuples representing links to other keys.

`RiakObject.indexes`

The set of secondary index entries, consisting of index-name/value tuples

`RiakObject.add_index` (*field*, *value*)

Tag this object with the specified field/value pair for indexing.

Parameters

- **field** (*string*) – The index field.
- **value** (*string or integer*) – The index value.

Return type *RiakObject*

`RiakObject.remove_index` (*field=None*, *value=None*)

Remove the specified field/value pair as an index on this object.

Parameters

- **field** (*string*) – The index field.
- **value** (*string or integer*) – The index value.

Return type *RiakObject*

`RiakObject.set_index` (*field*, *value*)

Works like *add_index()*, but ensures that there is only one index on given field. If other found, then removes it first.

Parameters

- **field**(*string*) – The index field.
- **value**(*string or integer*) – The index value.

Return type *RiakObject*

`RiakObject.add_link(obj, tag=None)`

Add a link to a *RiakObject*.

Parameters

- **obj**(*mixed*) – Either a *RiakObject* or 3 item link tuple consisting of (bucket, key, tag).
- **tag**(*string*) – Optional link tag. Defaults to bucket name. It is ignored if **obj** is a 3 item link tuple.

Return type *RiakObject*

5.3.4 Siblings

Because Riak’s consistency model is “eventual” (and not linearizable), there is no way for it to disambiguate writes that happen concurrently. The *Vector clock* helps establish a “happens after” relationships so that concurrent writes can be detected, but with the exception of *Data Types*, Riak has no way to determine which write has the correct value.

Instead, when *allow_mult* is *True*, Riak keeps all writes that appear to be concurrent. Thus, the contents of a key’s value may, in fact, be multiple values, which are called “siblings”. Siblings are modeled in *RiakContent* objects, which contain all of the same *Value and Metadata* methods and attributes as the parent object.

`RiakObject.siblings = []`

The list of sibling values contained in this object

```
class riak.content.RiakContent(robect, data=None, encoded_data=None, charset=None,
                               content_type='application/json', content_encoding=None,
                               last_modified=None, etag=None, usermeta=None, links=None,
                               indexes=None, exists=False)
```

The *RiakContent* holds the metadata and value of a single sibling within a *RiakObject*. *RiakObjects* that have more than one sibling are considered to be in conflict.

You do not typically have to create *RiakContent* objects yourself, but they will be created for you when *fetching* objects from Riak.

Note: The *Value and Metadata* accessors on *RiakObject* are actually proxied to the first sibling when the object has only one.

Conflicts and Resolvers

When an object is *not* in conflict, it has only one sibling. When it is in conflict, you will have to resolve the conflict before it can be written again. How you choose to resolve the conflict is up to you, but you can automate the process using a *resolver* function.

`riak.resolver.default_resolver(riak_object)`

The default conflict-resolution function, which does nothing. To implement a resolver, define a function that sets the *siblings* property on the passed *RiakObject* instance to a list containing a single *RiakContent* object.

Parameters **riak_object** (*RiakObject*) – an object-in-conflict that will be resolved

`riak.resolver.last_written_resolver(riak_object)`

A conflict-resolution function that resolves by selecting the most recently-modified sibling by timestamp.

Parameters `riak_object` (*RiakObject*) – an object-in-conflict that will be resolved

If you do not supply a resolver function, or your resolver leaves multiple siblings present, accessing the *Value* and *Metadata* will result in a *ConflictError* being raised.

exception `riak.ConflictError` (*message='Object in conflict'*)

Raised when an operation is attempted on a *RiakObject* that has more than one sibling.

5.4 Data Types

Traditionally all data stored in Riak was an opaque binary type. Then in version 1.4 came the introduction of a *counter*, the first Convergent Data Type supported in Riak. In Riak 2.0, several additional Data Types were introduced. Riak “knows” about these data types, and conflicting writes to them will converge automatically without presenting *sibling values* to the user.

Here is the list of current Data Types:

- *Counter* increments or decrements integer values
- *Set* allows you to store multiple distinct opaque binary values against a key
- *Map* is a nested, recursive struct, or associative array. Think of it as a container for composing ad hoc data structures from multiple Data Types. Inside a map you may store sets, counters, flags, registers, and even other maps
- *Register* stores binaries according to last-write-wins logic within *Map*
- *Flag* is similar to a boolean and also must be within *Map*

All Data Types must be stored in buckets bearing a *BucketType* that sets the *datatype* property to one of "counter", "set", or "map". Note that the bucket must have the *allow_mult* property set to `true`.

These Data Types are stored just like *RiakObjects*, so size constraints that apply to normal Riak values apply to Riak Data Types too.

An in-depth discussion of Data Types, also known as CRDTs, can be found at [Data Types](#).

Examples of using Data Types can be found at [Using Data Types](#).

5.4.1 Sending Operations

Riak Data Types provide a further departure from Riak’s usual operation, in that the API is operation-based. Rather than fetching the data structure, reconciling conflicts, mutating the result, and writing it back, you instead tell Riak what operations to perform on the Data Type. Here are some example operations:

- increment a *Counter* by 10
- add 'joe' to a *Set*
- remove the *Set* field called 'friends' from a *Map*
- enable the prepay *Flag* in a *Map*

Datatypes can be fetched and created just like *RiakObject* instances, using *RiakBucket.get* and *RiakBucket.new*, except that the bucket must belong to a bucket-type that has a valid datatype property. If we have a bucket-type named “social-graph” that has the datatype “set”, we would fetch a *Set* like so:

```
graph = client.bucket_type('social-graph')
graph.datatype # => 'set'
myfollowers = graph.bucket('followers').get('seancribbs')
# => a Set datatype
```

Once we have a datatype, we can stage operations against it and then send those operations to Riak:

```
myfollowers.add('javajolt')
myfollowers.discard('roach')
myfollowers.update()
```

While this looks in code very similar to manipulating *RiakObject* instances, only mutations are enqueued locally, not the new value.

5.4.2 Context and Observed-Remove

In order for Riak Data Types to behave well, you must have an opaque context received from a read when you:

- *disable* a *Flag* (set it to false)
- remove a field from a *Map*
- *remove* an element from a *Set*

The basic rule is “you cannot remove something you haven’t seen”, and the context tells Riak what you’ve actually seen, similar to the *Vector clock* on *RiakObject*. The Python client handles opaque contexts for you transparently as long as you fetch before performing one of these actions.

5.4.3 Datatype abstract class

class `riak.datatypes.Datatype` (*bucket=None, key=None, value=None, context=None*)

Base class for all convergent datatype wrappers. You will not use this class directly, but it does define some methods are common to all datatype wrappers.

value

The pure, immutable value of this datatype, as a Python value, which is unique for each datatype.

NB: Do not use this property to mutate data, as it will not have any effect. Use the methods of the individual type to effect changes. This value is guaranteed to be independent of any internal data representation.

context

The opaque context for this type, if it was previously fetched.

Return type str

modified

Whether this datatype has staged local modifications.

Return type bool

Persistence methods

Datatype.**reload** (***params*)

Reloads the datatype from Riak.

Parameters

- *r* (*integer, string, None*) – the read quorum

- **pr** (*integer*, *string*, *None*) – the primary read quorum
- **basic_quorum** (*bool*) – whether to use the “basic quorum” policy for not-found
- **notfound_ok** (*bool*) – whether to treat not-found responses as successful
- **timeout** (*int*) – a timeout value in milliseconds
- **include_context** (*bool*) – whether to return the opaque context as well as the value, which is useful for removal operations on sets and maps

Return type *Datatype*

`Datatype.update (**params)`

Sends locally staged mutations to Riak.

Parameters

- **w** (*integer*) – W-value, wait for this many partitions to respond before returning to client.
- **dw** (*integer*) – DW-value, wait for this many partitions to confirm the write before returning to client.
- **pw** (*integer*) – PW-value, require this many primary partitions to be available before performing the put
- **return_body** (*bool*) – if the newly stored object should be retrieved, defaults to True
- **include_context** (*bool*) – whether to return the new opaque context when *return_body* is True
- **timeout** (*int*) – a timeout value in milliseconds

Return type a subclass of *Datatype*

`Datatype.store (**params)`

This is an alias for *update()*.

`Datatype.delete (**params)`

Deletes the datatype from Riak. See *RiakClient.delete()* for options.

`Datatype.clear()`

Removes all locally staged mutations.

5.4.4 Counter

class `riak.datatypes.Counter` (*bucket=None*, *key=None*, *value=None*, *context=None*)

A convergent datatype that represents a counter which can be incremented or decremented. This type can stand on its own or be embedded within a *Map*.

`Counter.value`

The current value of the counter.

Return type `int`

`Counter.increment (amount=1)`

Increments the counter by one or the given amount.

Parameters **amount** (*int*) – the amount to increment the counter

`Counter.decrement (amount=1)`

Decrements the counter by one or the given amount.

Parameters **amount** (*int*) – the amount to decrement the counter

5.4.5 Set

class `riak.datatypes.Set` (*bucket=None, key=None, value=None, context=None*)

A convergent datatype representing a Set with observed-remove semantics. Currently strings are the only supported value type. Example:

```
myset.add('barista')
myset.add('roaster')
myset.add('brewer')
```

Likewise they can simply be removed:

```
myset.discard('barista')
```

This datatype also implements the [Set ABC](#), meaning it supports `len()`, `in`, and iteration.

Set.value

An immutable copy of the current value of the set.

Return type frozenset

Set.add (*element*)

Adds an element to the set.

Parameters **element** (*str*) – the element to add

Set.discard (*element*)

Removes an element from the set.

Parameters **element** (*str*) – the element to remove

5.4.6 Map

class `riak.datatypes.Map` (*bucket=None, key=None, value=None, context=None*)

A convergent datatype that acts as a key-value datastructure. Keys are pairs of (*name*, *datatype*) where *name* is a string and *datatype* is the datatype name. Values are other convergent datatypes, represented by any concrete type in this module.

You cannot set values in the map directly (it does not implement `__setitem__`), but you may add new empty values or access non-existing values directly via bracket syntax. If a key is not in the original value of the map when accessed, fetching the key will cause its associated value to be created.:

```
map[('name', 'register')]
```

Keys and their associated values may be deleted from the map as you would in a dict:

```
del map[('emails', 'set')]
```

Convenience accessors exist that partition the map's keys by datatype and implement the Mapping behavior as well as supporting deletion:

```
map.sets['emails']
map.registers['name']
del map.counters['likes']
```

Map.value

Returns a copy of the original map's value. Nested values are pure Python values as returned by `Datatype.value` from the nested types.

Return type dict

Map.**counters**

Filters keys in the map to only those of counter types. Example:

```
map.counters['views'].increment()
del map.counters['points']
```

Map.**flags**

Filters keys in the map to only those of flag types. Example:

```
map.flags['confirmed'].enable()
del map.flags['attending']
```

Map.**maps**

Filters keys in the map to only those of map types. Example:

```
map.maps['emails'].registers['home'].set("user@example.com")
del map.maps['spam']
```

Map.**registers**

Filters keys in the map to only those of register types. Example:

```
map.registers['username'].set_value("riak-user")
del map.registers['access_key']
```

Map.**sets**

Filters keys in the map to only those of set types. Example:

```
map.sets['friends'].add("brett")
del map.sets['favorites']
```

5.4.7 Map-only datatypes

Two of the new Data Types may only be embedded in *Map* objects (in addition to *Map* itself):

5.4.8 Register

class riak.datatypes.**Register** (*bucket=None, key=None, value=None, context=None*)

A convergent datatype that represents an opaque string that is set with last-write-wins semantics, and may only be embedded in *Map* instances.

Register.**value**

Returns a copy of the original value of the register.

Return type str

Register.**assign** (*new_value*)

Assigns a new value to the register.

Parameters **new_value** (*str*) – the new value for the register

5.4.9 Flag

class riak.datatypes.**Flag** (*bucket=None, key=None, value=None, context=None*)

A convergent datatype that represents a boolean value that can be enabled or disabled, and may only be embedded in *Map* instances.

`Flag.value`

The current value of the flag.

Return type bool, None

`Flag.enable()`

Turns the flag on, effectively setting its value to `True`.

`Flag.disable()`

Turns the flag off, effectively setting its value to `False`.

5.5 Query Methods

Although most operations you will do involve directly interacting with known buckets and keys, there are additional ways to get information out of Riak.

5.5.1 Secondary Indexes

Objects can be *tagged* with *secondary index entries*. Those entries can then be queried over *the bucket* for equality or across ranges.:

```
bucket = client.bucket("index_test")

# Tag an object with indexes and save
sean = bucket.new("seancribbs")
sean.add_index("fname_bin", "Sean")
sean.add_index("byear_int", 1979)
sean.store()

# Performs an equality query
seans = bucket.get_index("fname_bin", "Sean")

# Performs a range query
eighties = bucket.get_index("byear_int", 1980, 1989)
```

Secondary indexes are also available via *MapReduce*.

Streaming and Paginating Indexes

Sometimes the number of results from such a query is too great to process in one payload, so you can also *stream the results*:

```
for keys in bucket.stream_index("bmonth_int", 1):
    # keys is a list of matching keys
    print(keys)
```

Both the regular `get_index()` method and the `stream_index()` method allow you to return the index entry along with the matching key as tuples using the `return_terms` option:

```
bucket.get_index("byear_int", 1970, 1990, return_terms=True)
# => [(1979, 'seancribbs')]
```

You can also limit the number of results using the `max_results` option, which enables pagination:

```
results = bucket.get_index("fname_bin", "S", "T", max_results=20)
```

Optionally you can use `paginate_index()` or `paginate_stream_index()` to create a generator of paged results:

```
for page in bucket.paginate_stream_index("maestro_bin", "Cribbs"):
    for key in page:
        do_something(key)
    page.close()
```

All of these features are implemented using the `IndexPage` class, which emulates a list but also supports streaming and capturing the *continuation*, which is a sort of pointer to the next page of results:

```
# Detect whether there are more results
if results.has_next_page():

    # Fetch the next page of results manually
    more = bucket.get_index("fname_bin", "S", "T", max_results=20,
                           continuation=results.continuation)

    # Fetch the next page of results automatically
    more = results.next_page()
```

```
class riak.client.index_page.IndexPage(client, bucket, index, startkey, endkey, return_terms,
                                       max_results, term_regex)
```

Encapsulates a single page of results from a secondary index query, with the ability to iterate over results (if not streamed), capture the page marker (continuation), and automatically fetch the next page.

While users will interact with this object, it will be created automatically by the client and does not need to be instantiated elsewhere.

continuation = None

The opaque page marker that is used when fetching the next chunk of results. The user can simply call `next_page()` to do so, or pass this to the `get_index()` method using the `continuation` option.

has_next_page()

Whether there is another page available, i.e. the response included a continuation.

next_page(timeout=None, stream=None)

Fetches the next page using the same parameters as the original query.

Note that if streaming was used before, it will be used again unless overridden.

Parameters

- **stream** (*boolean*) – whether to enable streaming. *True* enables, *False* disables, *None* uses previous value.
- **timeout** (*int*) – a timeout value in milliseconds, or ‘infinity’

__eq__ (*other*)

An `IndexPage` can pretend to be equal to a list when it has captured results by simply comparing the internal results to the passed list. Otherwise the other object needs to be an equivalent `IndexPage`.

__iter__ ()

Emulates the iterator interface. When streaming, this means delegating to the stream, otherwise iterating over the existing result set.

__getitem__ (*index*)

Fetches an item by index from the captured results.

5.5.2 MapReduce

RiakMapReduce allows you to construct query-processing jobs that are performed mostly in-parallel around the Riak cluster. You can think of it as a pipeline, where inputs are fed in one end, they pass through a number of map and reduce phases, and then are returned to the client.

Constructing the query

class `riak.mapreduce.RiakMapReduce` (*client*)

The `RiakMapReduce` object allows you to build up and run a map/reduce operation on Riak. Most methods return the object on which it was called, modified with new information, so you can chain calls together to build the job.

Construct a Map/Reduce object.

Parameters `client` (*RiakClient*) – the client that will perform the query

Inputs

The first step is to identify the inputs that should be processed. They can be:

1. An entire *bucket*
2. An entire bucket, with the *keys filtered by criteria*
3. A *list of bucket/key pairs* or bucket/key/data triples
4. A *fulltext search query*
5. A *secondary-index query*

Adding inputs always returns the `RiakMapReduce` object so that you can chain the construction of the query job.

`RiakMapReduce.add_bucket` (*bucket*, *bucket_type=None*)

Adds all keys in a bucket to the inputs.

Parameters

- **bucket** (*string*) – the bucket
- **bucket_type** (*string*, *None*) – Optional name of a bucket type

Return type *RiakMapReduce*

`RiakMapReduce.add_key_filters` (*key_filters*)

Adds key filters to the inputs.

Parameters **key_filters** (*list*) – a list of filters

Return type *RiakMapReduce*

`RiakMapReduce.add_key_filter` (**args*)

Add a single key filter to the inputs.

Parameters **args** (*list*) – a filter

Return type *RiakMapReduce*

`RiakMapReduce.add` (*arg1*, *arg2=None*, *arg3=None*, *bucket_type=None*)

Add inputs to a map/reduce operation. This method takes three different forms, depending on the provided inputs. You can specify either a `RiakObject`, a string bucket name, or a bucket, key, and additional arg.

Parameters

- **arg1** (*RiakObject*, *string*) – the object or bucket to add
- **arg2** (*string*, *list*, *None*) – a key or list of keys to add (if a bucket is given in arg1)
- **arg3** (*string*, *list*, *dict*, *None*) – key data for this input (must be convertible to JSON)
- **bucket_type** (*string*, *None*) – Optional name of a bucket type

Return type *RiakMapReduce*

`RiakMapReduce.add_object(obj)`

Adds a *RiakObject* to the inputs.

Parameters **obj** (*RiakObject*) – the object to add

Return type *RiakMapReduce*

`RiakMapReduce.add_bucket_key_data(bucket, key, data, bucket_type=None)`

Adds a bucket/key/keydata triple to the inputs.

Parameters

- **bucket** (*string*) – the bucket
- **key** (*string*) – the key or list of keys
- **data** (*string*, *list*, *dict*, *None*) – the key-specific data
- **bucket_type** (*string*, *None*) – Optional name of a bucket type

Return type *RiakMapReduce*

`RiakMapReduce.search(index, query)`

Begin a map/reduce operation using a Search. This command will return an error unless executed against a Riak Search cluster.

Parameters

- **index** (*string*) – The Solr index used in the search
- **query** (*string*) – The search query

Return type *RiakMapReduce*

`RiakMapReduce.index(bucket, index, startkey, endkey=None, bucket_type=None)`

Begin a map/reduce operation using a Secondary Index query.

Parameters

- **bucket** (*string*) – The bucket over which to perform the query
- **index** (*string*) – The index to use for query
- **startkey** (*string*, *integer*) – The start key of index range, or the value which all entries must equal
- **endkey** (*string*, *integer*, *None*) – The end key of index range (if doing a range query)
- **bucket_type** (*string*, *None*) – Optional name of a bucket type

Return type *RiakMapReduce*

`class riak.mapreduce.RiakKeyFilter(*args)`

A helper class for building up lists of key filters. Unknown methods are treated as filters to be added; & and | create conjunctions and disjunctions, respectively. + concatenates filters.

Example:

```
f1 = RiakKeyFilter().starts_with('2005')
f2 = RiakKeyFilter().ends_with('-01')
f3 = f1 & f2
print(f3)
# => [['and', [['starts_with', '2005']], [['ends_with', '-01']]]]
```

Parameters `args (list)` – a list of arguments to be treated as a filter.

Phases

The second step is to add processing phases to the query. `map` phases load and process individual keys, returning one or more results, while `reduce` phases operate over collections of results from previous phases. `link` phases are a special type of map phase that extract matching *links* from the object, usually so they can be used in a subsequent map phase.

Any number of phases can return results directly to the client by passing `keep=True`.

`RiakMapReduce.map (function, options=None)`

Add a map phase to the map/reduce operation.

Parameters

- **function** (*string, list*) – Either a named Javascript function (ie: ‘`Riak.mapValues`’), or an anonymous javascript function (ie: ‘`function(...) ...`’ or an array [`erlang_module`, ‘`function`’]).
- **options** (*dict*) – phase options, containing ‘`language`’, ‘`keep`’ flag, and/or ‘`arg`’.

Return type *RiakMapReduce*

`RiakMapReduce.reduce (function, options=None)`

Add a reduce phase to the map/reduce operation.

Parameters

- **function** (*string, list*) – Either a named Javascript function (ie. ‘`Riak.reduceSum`’), or an anonymous javascript function (ie: ‘`function(...) { ... }`’ or an array [`erlang_module`, ‘`function`’]).
- **options** – phase options, containing ‘`language`’, ‘`keep`’ flag, and/or ‘`arg`’.

Return type *RiakMapReduce*

`RiakMapReduce.link (bucket='_', tag='_', keep=False)`

Add a link phase to the map/reduce operation.

Parameters

- **bucket** (*string*) – Bucket name (default ‘`_`’, which means all buckets)
- **tag** (*string*) – Tag (default ‘`_`’, which means any tag)
- **keep** (*boolean*) – Flag whether to keep results from this stage in the map/reduce. (default `False`, unless this is the last step in the phase)

Return type *RiakMapReduce*

`class riak.mapreduce.RiakMapReducePhase (type, function, language, keep, arg)`

The `RiakMapReducePhase` holds information about a Map or Reduce phase in a `RiakMapReduce` operation.

Normally you won't need to use this object directly, but instead call methods on `RiakMapReduce` objects to add instances to the query.

Construct a `RiakMapReducePhase` object.

Parameters

- **type** (*string*) – the phase type - 'map', 'reduce', 'link'
- **function** (*string*, *list*) – the function to execute
- **language** (*string*) – 'javascript' or 'erlang'
- **keep** (*boolean*) – whether to return the output of this phase in the results.
- **arg** (*string*, *dict*, *list*) – Additional static value to pass into the map or reduce function.

class `riak.mapreduce.RiakLinkPhase` (*bucket*, *tag*, *keep*)

The `RiakLinkPhase` object holds information about a Link phase in a map/reduce operation.

Normally you won't need to use this object directly, but instead call `RiakMapReduce.link()` on `RiakMapReduce` objects to add instances to the query.

Construct a `RiakLinkPhase` object.

Parameters

- **bucket** (*string*) –
– The bucket name
- **tag** (*string*) – The tag
- **keep** (*boolean*) – whether to return results of this phase.

Phase shortcuts

A number of commonly-used phases are also available as shortcut methods:

`RiakMapReduce.map_values` (*options=None*)

Adds the Javascript built-in `Riak.mapValues` to the query as a map phase.

Parameters *options* (*dict*) – phase options, containing 'language', 'keep' flag, and/or 'arg'.

`RiakMapReduce.map_values_json` (*options=None*)

Adds the Javascript built-in `Riak.mapValuesJson` to the query as a map phase.

Parameters *options* (*dict*) – phase options, containing 'language', 'keep' flag, and/or 'arg'.

`RiakMapReduce.reduce_sum` (*options=None*)

Adds the Javascript built-in `Riak.reduceSum` to the query as a reduce phase.

Parameters *options* (*dict*) – phase options, containing 'language', 'keep' flag, and/or 'arg'.

`RiakMapReduce.reduce_min` (*options=None*)

Adds the Javascript built-in `Riak.reduceMin` to the query as a reduce phase.

Parameters *options* (*dict*) – phase options, containing 'language', 'keep' flag, and/or 'arg'.

`RiakMapReduce.reduce_max` (*options=None*)

Adds the Javascript built-in `Riak.reduceMax` to the query as a reduce phase.

Parameters *options* (*dict*) – phase options, containing 'language', 'keep' flag, and/or 'arg'.

`RiakMapReduce.reduce_sort (js_cmp=None, options=None)`

Adds the Javascript built-in `Riak.reduceSort` to the query as a reduce phase.

Parameters

- **js_cmp** (*string*) – A Javascript comparator function as specified by `Array.sort()`
- **options** (*dict*) – phase options, containing ‘language’, ‘keep’ flag, and/or ‘arg’.

`RiakMapReduce.reduce_numeric_sort (options=None)`

Adds the Javascript built-in `Riak.reduceNumericSort` to the query as a reduce phase.

Parameters **options** (*dict*) – phase options, containing ‘language’, ‘keep’ flag, and/or ‘arg’.

`RiakMapReduce.reduce_limit (limit, options=None)`

Adds the Javascript built-in `Riak.reduceLimit` to the query as a reduce phase.

Parameters

- **limit** (*integer*) – the maximum number of results to return
- **options** (*dict*) – phase options, containing ‘language’, ‘keep’ flag, and/or ‘arg’.

`RiakMapReduce.reduce_slice (start, end, options=None)`

Adds the Javascript built-in `Riak.reduceSlice` to the query as a reduce phase.

Parameters

- **start** (*integer*) – the beginning of the slice
- **end** (*integer*) – the end of the slice
- **options** (*dict*) – phase options, containing ‘language’, ‘keep’ flag, and/or ‘arg’.

`RiakMapReduce.filter_not_found (options=None)`

Adds the Javascript built-in `Riak.filterNotFound` to the query as a reduce phase.

Parameters **options** (*dict*) – phase options, containing ‘language’, ‘keep’ flag, and/or ‘arg’.

Execution

Query results can either be executed in one round-trip, or streamed back to the client. The format of results will depend on the structure of the map and reduce phases the query contains.

`RiakMapReduce.run (timeout=None)`

Run the map/reduce operation synchronously. Returns a list of results, or a list of links if the last phase is a link phase. Shortcut for `riak.client.RiakClient.mapred()`.

Parameters **timeout** (*integer, None*) – Timeout in milliseconds

Return type list

`RiakMapReduce.stream (timeout=None)`

Streams the MapReduce query (returns an iterator). Shortcut for `riak.client.RiakClient.stream_mapred()`.

Parameters **timeout** (*integer*) – Timeout in milliseconds

Return type iterator that yields (phase_num, data) tuples

Shortcut constructors

RiakObject contains some shortcut methods that make it more convenient to begin constructing *RiakMapReduce* queries.

`RiakObject.add(arg1, arg2=None, arg3=None, bucket_type=None)`

Start assembling a Map/Reduce operation. A shortcut for `add()`.

Parameters

- **arg1** (*RiakObject*, *string*) – the object or bucket to add
- **arg2** (*string*, *list*, *None*) – a key or list of keys to add (if a bucket is given in arg1)
- **arg3** (*string*, *list*, *dict*, *None*) – key data for this input (must be convertible to JSON)
- **bucket_type** (*string*, *None*) – Optional name of a bucket type

Return type *RiakMapReduce*

`RiakObject.link(*args)`

Start assembling a Map/Reduce operation. A shortcut for `link()`.

Return type *RiakMapReduce*

`RiakObject.map(*args)`

Start assembling a Map/Reduce operation. A shortcut for `map()`.

Return type *RiakMapReduce*

`RiakObject.reduce(*args)`

Start assembling a Map/Reduce operation. A shortcut for `reduce()`.

Return type *RiakMapReduce*

5.5.3 Riak Search 2.0 (Yokozuna)

With Riak 2.0 came the introduction of **Riak Search 2.0**, a.k.a *Yokozuna* (the top rank in sumo). Riak Search 2.0 is an integration of Solr (for indexing and querying) and Riak (for storage and distribution). It allows for distributed, scalable, fault-tolerant, transparent indexing and querying of Riak values. After connecting a bucket (or bucket type) to a [Apache Solr](#) index, you simply write values (such as JSON, XML, plain text, Data Types, etc.) into Riak as normal, and then query those indexed values using the Solr API. Unlike traditional Riak data, however, Solr needs to know the format of the stored data so it can index it. Solr is a document-based search engine so it treats each value stored in Riak as a document.

Creating a schema

The first thing which needs to be done is to define a Solr schema for your data. Riak Search comes bundled with a default schema named `_yz_default`. It defaults to many dynamic field types, where the suffix defines its type. This is an easy path to start development, but we recommend in production that you define your own schema.

You can find information about defining your own schema at [Search Schema](#), with a short section dedicated to the [default schema](#).

Here is a brief example of creating a custom schema with `create_search_schema()`:

```
content = """<?xml version="1.0" encoding="UTF-8" ?>
<schema name="test" version="1.5">
<fields>
```

```
<field name="_yz_id" type="_yz_str" indexed="true" stored="true"
  multiValued="false" required="true" />
<field name="_yz_ed" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_pn" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_fpn" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_vtag" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_rk" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_rb" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_rt" type="_yz_str" indexed="true" stored="true"
  multiValued="false" />
<field name="_yz_err" type="_yz_str" indexed="true"
  multiValued="false" />
</fields>
<uniqueKey>_yz_id</uniqueKey>
<types>
  <fieldType name="_yz_str" class="solr.StrField"
    sortMissingLast="true" />
</types>
</schema>"""
schema_name = 'jalapeno'
client.create_search_schema(schema_name, content)
```

If you would like to retrieve the current XML Solr schema, `get_search_schema()` is available:

```
schema = client.get_search_schema('jalapeno')
```

Solr indexes

Once a schema has been created, then a Solr index must also be created. This index represents a collection of similar data that you use to perform queries. When creating an index with `create_search_index()`, you can optionally specify a schema. If you do not, the default schema will be used:

```
client.create_search_index('nacho')
```

Likewise you can specify a schema, e.g. the index "nacho" is associated with the schema "jalapeno":

```
client.create_search_index('nacho', 'jalapeno')
```

Just as easily you can delete an index with `delete_search_index()`:

```
client.delete_search_index('jalapeno')
```

A single index can be retrieved with `get_search_index()` or all of them with `list_search_indexes()`:

```
index = client.get_search_index('jalapeno')
name = index['name']
schema = index['schema']
indexes = client.list_search_indexes()
first_nval = indexes[0]['n_val']
```

Note: Note that index names may only be ASCII values from 32-127 (spaces, standard punctuation, digits and word

characters). This may change in the future to allow full unicode support.

More discussion about Riak Search 2.0 Indexes can be found at [Indexes](#).

Linking a bucket type to an index

The last step to setting up Riak Search 2.0 is to link a Bucket Type to a Solr index. This lets Riak know when to index values. This can be done via the command line:

```
riak-admin bucket-type create spicy '{"props":{"search_index":"jalapeno"}}'
riak-admin bucket-type activate spicy
```

Or simply create an empty Bucket Type:

```
riak-admin bucket-type create spicy '{"props":{}}'
riak-admin bucket-type activate spicy
```

Then change the bucket properties on the associated bucket or Bucket Type:

```
b = client.bucket('peppers')
b.set_property('search_index', 'jalapeno')
btype = client.bucket_type('spicy')
btype.set_property('search_index', 'jalapeno')
```

Querying an index

Once the schema, index and bucket properties have all been properly configured, adding data is as simple as writing to Riak. Solr is automatically updated.

To query, on the other hand, is as easy as writing Solr queries. This allows for the full use of existing Solr tools as well as its rich semantics.

Here is a brief example of loading and querying data::

```
bucket = self.client.bucket('peppers')
bucket.new("bell", {"name_s": "bell", "scoville_low_i": 0,
                    "scoville_high_i": 0}).store()
bucket.new("anaheim", {"name_s": "anaheim", "scoville_low_i": 1000,
                       "scoville_high_i": 2500}).store()
bucket.new("chipotle", {"name_s": "chipotle", "scoville_low_i": 3500,
                        "scoville_high_i": 10000}).store()
bucket.new("serrano", {"name_s": "serrano", "scoville_low_i": 10000,
                       "scoville_high_i": 23000}).store()
bucket.new("habanero", {"name_s": "habanero", "scoville_low_i": 100000,
                        "scoville_high_i": 350000}).store()
results = bucket.search("name_s:/c.*/", index='jalapeno')
# Yields single document 'chipotle'
print(results['docs'][0]['name_s'])
results = bucket.search("scoville_high_i:[20000 TO 500000]")
# Yields two documents
for result in results['docs']:
    print(result['name_s'])
results = bucket.search('name_s:*', index='jalapeno',
                        sort="scoville_low_i desc")
# Yields all documents, sorted in descending order. We take the top one
print("The hottest pepper is {}".format(results['docs'][0]['name_s']))
```

The results returned by `search()` is a dictionary with lots of search metadata like the number of results, the maximum **Lucene Score** as well as the matching documents.

When querying on *Data Types* the datatype is the name of the field used in Solr since they do not fit into the default schema, e.g.:

```
riak-admin bucket-type create visitors '{"props":{"datatype": "counter"}}'
riak-admin bucket-type activate visitors
```

```
client.create_search_index('website')
bucket = client.bucket_type('visitors').bucket('hits')
bucket.set_property('search_index', 'website')

site = bucket.new('bbc.co.uk')
site.increment(80)
site.store()
site = bucket.new('cnn.com')
site.increment(150)
site.store()
site = bucket.new('abc.net.au')
site.increment(24)
site.store()

results = bucket.search("counter:[10 TO *]", index='website',
                        sort="counter desc", rows=5)

# Assume you have a bucket-type named "profiles" that has datatype
# "map". Let's create and search an index containing maps.
client.create_search_index('user-profiles')
bucket = client.bucket_type('profiles').bucket('USA')
bucket.set_property('search_index', 'user-profiles')

brett = bucket.new()
brett.registers['fname'].assign("Brett")
brett.registers['lname'].assign("Hazen")
brett.sets['emails'].add('spam@basho.com')
brett.counters['visits'].increment()
brett.maps['pages'].counters['homepage'].increment()
brett.update()

# Note that the field name in the index/schema is the field name in
# the map joined with its type by an underscore. Deeply embedded
# fields are joined with their parent field names by an underscore.
results = bucket.search('lname_register:Hazen AND pages_map_homepage_counter:[1 TO *]',
                        index='user-profiles')
```

Details on querying Riak Search 2.0 can be found at [Querying](#).

5.6 Security

Riak 2.0 supports authentication and authorization over encrypted channels via OpenSSL. This is useful to prevent accidental collisions between environments (e.g., pointing application software under active development at the production cluster) and offers protection against some malicious attacks, although Riak still should not be exposed directly to any unsecured network.

Several important caveats when enabling security:

- There is no support yet for auditing. This is on the roadmap for a future release.

- Two deprecated features will not work if security is enabled: link walking and Riak Search 1.0.
- There are restrictions on Erlang modules exposed to MapReduce jobs when security is enabled.
- Enabling security requires applications be designed to transition gracefully based on the server response or applications will need to be halted before security is enabled and brought back online with support for the new security features.

5.6.1 Server Configuration

The server must first be configured to [enable security](#), users and [security sources](#) must be created, [permissions](#) applied and the correct certificates must be installed. An overview can be found at [Authentication and Authorization](#).

5.6.2 Client Configuration

Note: OpenSSL 1.0.1g or later (or patched version built after 2014-04-01) is required for [pyOpenSSL](#), which is used for secure transport in the Riak client. Earlier versions may not support TLS 1.2, the recommended security protocol.

On the client, simply create a [SecurityCreds](#) object with just a username, password and CA Certificate file. That would then need to be passed into the [RiakClient](#) initializer:

```
creds = SecurityCreds('riakuser',
                     'riakpass',
                     cacert_file='/path/to/ca.crt')
client = RiakClient(credentials=creds)
```

The `credentials` argument of a [RiakClient](#) constructor is a [SecurityCreds](#) object. If you specify a dictionary instead, it will be turned into this type:

```
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'cacert_file': '/path/to/ca.crt'}
client = RiakClient(credentials=creds)
```

Note: A Certifying Authority (CA) Certificate must always be supplied to [SecurityCreds](#) by specifying the path to a CA certificate file via the `cacert_file` argument or by setting the `cacert` argument to an [OpenSSL.crypto.X509](#) object. This mitigates MITM (man-in-the-middle) attacks by ensuring correct certificate validation.

5.6.3 Authentication Types

Trust and PAM Authentication

The most basic authentication would be [Trust-based Authentication](#) which is done exclusively on the server side by adding the appropriate `trust` security source:

```
riak-admin security add-source all 127.0.0.1/32 trust
```

[PAM-based Authentication](#) is another server-side solution which can be added by a `pam` security source with the name of the service:

```
riak-admin security add-source all 127.0.0.1/32 pam service=riak_pam
```

Even if you are using Trust authentication or the PAM module doesn't require a password, you must supply one to the client API. From the client's perspective, these are equivalent to Password authentication.

Password Authentication

The next level of security would be simply a username and password for [Password-based Authentication](#). The server needs to first have a user and a password security source:

```
riak-admin security add-user riakuser password=captheorem4life
riak-admin security add-source riakuser 127.0.0.1/32 password
```

On the client, simply create a *SecurityCreds* object or dict with just a username and password. That would then need to be passed into the *RiakClient* initializer:

```
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'cacert_file': '/path/to/ca.crt'}
client = RiakClient(credentials=creds)
myBucket = client.bucket('test')
vall = "#SeanCribbsHoldingThings"
key1 = myBucket.new('hashtag', data=vall)
key1.store()
```

Client Certificate Authentication

If you are using the **Protocol Buffers** transport you could also add a layer of security by using [Certificate-based Authentication](#). This time the server requires a certificate security source:

```
riak-admin security add-source riakuser 127.0.0.1/32 certificate
```

When the certificate source is used, the Riak username must match the common name, aka CN, that you specified when you generated your certificate. You can add a certificate source to any number of clients.

The *SecurityCreds* must then include the include a client certificate file and a private key file, too:

```
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'cacert_file': '/path/to/ca.crt',
        'cert_file': '/path/to/client.crt',
        'pkey_file': '/path/to/client.key'}
```

Note: Username and password are still required for certificate-based authentication, although the password is ignored.

Optionally, the certificate or private key may be supplied as a string:

```
with open('/path/to/client.key', 'r') as f:
    preloaded_pkey = f.read()
with open('/path/to/client.crt', 'r') as f:
    preloaded_cert = f.read()
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'cert': preloaded_cert,
        'pkey': prelocated_pkey}
```

5.6.4 Additional options

Certificate revocation lists

Another security option available is a Certificate Revocation List (CRL). It lists server certificates which, for whatever reason, are no longer valid. For example, it is discovered that the certificate authority (CA) had improperly issued a certificate, or if a private-key is thought to have been compromised. The most common reason for revocation is the user no longer being in sole possession of the private key (e.g., the token containing the private key has been lost or stolen):

```
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'cacert_file': '/path/to/ca.crt',
        'crl_file': '/path/to/server.crl'}
```

Cipher options

The last interesting setting on `SecurityCreds` is the `ciphers` option which is a colon-delimited list of supported ciphers for encryption:

```
creds = {'username': 'riakuser',
        'password': 'riakpass',
        'ciphers': 'ECDHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA' }
```

A more detailed discussion can be found at [Security Ciphers](#).

5.6.5 SecurityCreds object

```
class riak.security.SecurityCreds (username=None, password=None, pkey_file=None,
                                   pkey=None, cert_file=None, cert=None, cacert_file=None,
                                   cacert=None, crl_file=None, crl=None, ciphers=None,
                                   ssl_version=3)
```

Container class for security-related settings

Parameters

- **username** (*str*) – Riak Security username
- **password** (*str*) – Riak Security password
- **pkey_file** (*str*) – Full path to security key file
- **key** (`OpenSSL.crypto.PKey`) – Loaded security key file
- **cert_file** (*str*) – Full path to certificate file
- **cert** (`OpenSSL.crypto.X509`) – Loaded client certificate
- **cacert_file** (*str*) – Full path to CA certificate file
- **cacert** (`OpenSSL.crypto.X509`) – Loaded CA certificate
- **crl_file** (*str*) – Full path to revoked certificates file
- **crl** (`OpenSSL.crypto.CRL`) – Loaded revoked certificates list
- **ciphers** (*str*) – List of supported SSL ciphers
- **ssl_version** (*int*) – OpenSSL security version

username
Riak Username
Return type str

password
Riak Password
Return type str

cacert
Certifying Authority (CA) Certificate
Return type OpenSSL.crypto.X509

crl
Certificate Revocation List
Return type OpenSSL.crypto.CRL

cert
Client Certificate
Return type OpenSSL.crypto.X509

pkey
Client Private key
Return type OpenSSL.crypto.PKey

ciphers
Colon-delimited list of supported ciphers
Return type str

ssl_version
SSL/TLS Protocol to use
Return type an int constant from OpenSSL, like OpenSSL.SSL.TLSv1_2_METHOD

5.7 Advanced Usage & Internals

This page contains documentation for aspects of library internals that you will rarely need to interact with, but are important for understanding how it works and development purposes.

5.7.1 Connection pool

exception riak.transports.pool.**BadResource**

Users of a *Pool* should raise this error when the pool resource currently in-use is bad and should be removed from the pool.

class riak.transports.pool.**Resource** (*obj, pool*)

A member of the *Pool*, a container for the actual resource being pooled and a marker for whether the resource is currently claimed.

Creates a new Resource, wrapping the passed object as the pooled resource.

Parameters *obj* (*object*) – the resource to wrap

release ()

Releases this resource back to the pool it came from.

claimed = None

Whether the resource is currently in use.

object = None

The wrapped pool resource.

pool = None

The pool that this resource belongs to.

class `riak.transports.pool.Pool`

A thread-safe, reentrant resource pool, ported from the “Innertube” Ruby library. Pool should be subclassed to implement the `create_resource` and `destroy_resource` functions that are responsible for creating and cleaning up the resources in the pool, respectively. Claiming a resource of the pool for a block of code is done using a with statement on the transaction method. The transaction method also allows filtering of the pool and supplying a default value to be used as the resource if no resources are free.

Example:

```
from riak.Pool import Pool, BadResource
class ListPool(Pool):
    def create_resource(self):
        return []

    def destroy_resource(self):
        # Lists don't need to be cleaned up
        pass

pool = ListPool()
with pool.transaction() as resource:
    resource.append(1)
with pool.transaction() as resource2:
    print(repr(resource2)) # should be [1]
```

Creates a new Pool. This should be called manually if you override the `__init__()` method in a subclass.

acquire (*_filter=None, default=None*)

Claims a resource from the pool for manual use. Resources are created as needed when all members of the pool are claimed or the pool is empty. Most of the time you will want to use `transaction()`.

Parameters

- **_filter** (*callable*) – a filter that can be used to select a member of the pool
- **default** – a value that will be used instead of calling `create_resource()` if a new resource needs to be created

Return type *Resource*

clear ()

Removes all resources from the pool, calling `delete_resource()` with each one so that the resources are cleaned up.

create_resource ()

Implemented by subclasses to allocate a new resource for use in the pool.

delete_resource (*resource*)

Deletes the resource from the pool and destroys the associated resource. Not usually needed by users of the pool, but called internally when `BadResource` is raised.

Parameters **resource** (*Resource*) – the resource to remove

destroy_resource (*obj*)

Called when removing a resource from the pool so that it can be cleanly deallocated. Subclasses should

implement this method if additional cleanup is needed beyond normal GC. The default implementation is a no-op.

Parameters `obj` – the resource being removed

release (*resource*)

Returns a resource to the pool. Most of the time you will want to use `transaction()`, but if you use `acquire()`, you must release the acquired resource back to the pool when finished. Failure to do so could result in deadlock.

Parameters `resource` – Resource

transaction (*_filter=None, default=None*)

Claims a resource from the pool for use in a thread-safe, reentrant manner (as part of a with statement). Resources are created as needed when all members of the pool are claimed or the pool is empty.

Parameters

- **_filter** (*callable*) – a filter that can be used to select a member of the pool
- **default** – a value that will be used instead of calling `create_resource()` if a new resource needs to be created

class `riak.transports.pool.PoolIterator` (*pool*)

Iterates over a snapshot of the pool in a thread-safe manner, eventually touching all resources that were known when the iteration started.

Note that if claimed resources are not released for long periods, the iterator may hang, waiting for those last resources to be released. The iteration and pool functionality is only meant to be used internally within the client, and resources will be claimed per client operation, making this an unlikely event (although still possible).

5.7.2 Retry logic

class `riak.client.transport.RiakClientTransport`

Methods for RiakClient related to transport selection and retries.

_acquire ()

Acquires a connection from the default pool.

_choose_pool (*protocol=None*)

Selects a connection pool according to the default protocol and the passed one.

Parameters `protocol` (*string*) – the protocol to use

Return type *Pool*

_transport ()

Yields a single transport to the caller from the default pool, without retries.

_with_retries (*pool, fn*)

Performs the passed function with retries against the given pool.

Parameters

- **pool** (*Pool*) – the connection pool to use
- **fn** (*function*) – the function to pass a transport

retry_count (*retries*)

Modifies the number of retries for the scope of the with statement (in the current thread).

Example:

```
with client.retry_count(10):
    client.ping()
```

retries

The number of times retryable operations will be attempted before raising an exception to the caller. Defaults to 3.

Note This is a thread-local for safety and operation-specific modification. To change the default globally, modify `riak.client.transport.DEFAULT_RETRY_COUNT`.

`riak.client.transport._is_retryable(error)`

Determines whether a given error is retryable according to the exceptions allowed to be retried by each transport.

Parameters `error` (*Exception*) – the error to check

Return type boolean

`riak.client.transport.retryable(fn, protocol=None)`

Wraps a client operation that can be retried according to the set `RiakClient.retries`. Used internally.

`riak.client.transport.retryableHttpOnly(fn)`

Wraps a retryable client operation that is only valid over HTTP. Used internally.

5.7.3 Multiget

`riak.client.multiget.POOL_SIZE = 4`

The default size of the worker pool, either based on the number of CPUS or defaulting to 6

class `riak.client.multiget.Task(client, outq, bucket_type, bucket, key, options)`

A namedtuple for tasks that are fed to workers in the multiget pool.

class `riak.client.multiget.MultiGetPool(size=4)`

Encapsulates a pool of fetcher threads. These threads can be used across many multi-get requests.

Parameters `size` (*int*) – the desired size of the worker pool

`__fetcher()`

The body of the multi-get worker. Loops until `__should_quit()` returns True, taking tasks off the input queue, fetching the object, and putting them on the output queue.

`__should_quit()`

Worker threads should exit when the stop flag is set and the input queue is empty. Once the stop flag is set, new enqueues are disallowed, meaning that the workers can safely drain the queue before exiting.

Return type bool

`enq(task)`

Enqueues a fetch task to the pool of workers. This will raise a `RuntimeError` if the pool is stopped or in the process of stopping.

Parameters `task` (*Task*) – the Task object

`start()`

Starts the worker threads if they are not already started. This method is thread-safe and will be called automatically when executing a MultiGet operation.

`stop()`

Signals the worker threads to exit and waits on them.

`stopped()`

Detects whether this pool has been stopped.

`riak.client.multiget.RIAK_MULTIGET_POOL = <riak.client.multiget.MultiGetPool object>`

The default pool is automatically created and stored in this constant.

`riak.client.multiget.multiget (client, keys, **options)`

Executes a parallel-fetch across multiple threads. Returns a list containing *RiakObject* or *Datatype* instances, or 4-tuples of bucket-type, bucket, key, and the exception raised.

If a pool option is included, the request will use the given worker pool and not the default *RIAK_MULTIGET_POOL*. This option will be passed by the client if the `multiget_pool_size` option was set on client initialization.

Parameters

- **client** (*RiakClient*) – the client to use
- **keys** (list of three-tuples -- bucket_type/bucket/key) – the keys to fetch in parallel
- **options** (dict) – request options to *RiakBucket.get*

Return type list

5.7.4 Datatypes

Datatype internals

`Datatype.to_op()`

Extracts the mutation operation from this datatype, if any. Each type must implement this method, returning the appropriate operation, or *None* if there is no queued mutation.

`Datatype._check_type (new_value)`

Checks that initial values of the type are appropriate. Each type must implement this method.

Return type bool

`Datatype._coerce_value (new_value)`

Coerces the input value into the internal representation for the type. Datatypes may override this method.

`Datatype._default_value()`

Returns what the initial value of an empty datatype should be.

`Datatype._post_init()`

Called at the end of `__init__()` so that subclasses can tweak their own setup without overriding the constructor.

`Datatype._require_context()`

Raises an exception if the context is not present

`Datatype.type_name = None`

The string “name” of this datatype. Each datatype should set this.

`Datatype._type_error_msg = ‘Invalid value type’`

The message included in the exception raised when the value is of incorrect type. See also `_check_type()`.

TypedMapView

`class riak.datatypes.map.TypedMapView (parent, datatype)`

Implements a sort of view over a *Map*, filtered by the embedded datatype.

__contains__ (*key*)
 Determines whether the given key with this view's datatype is in the parent *Map*.

__delitem__ (*key*)
 Removes the key with this view's datatype from the parent *Map*.

__getitem__ (*key*)
 Fetches an item from the parent *Map* scoped by this view's datatype.

Parameters *key* (*str*) – the key of the item

Return type *Datatype*

__iter__ ()
 Iterates over all keys in the *Map* scoped by this view's datatype.

__len__ ()
 Returns the number of keys in this map scoped by this view's datatype.

TYPES constant

`riak.datatypes.TYPES = {'map': <class 'riak.datatypes.map.Map'>, 'flag': <class 'riak.datatypes.flag.Flag'>, 'counter': <class 'riak.datatypes.counter.Counter'>}`

`dict()` -> new empty dictionary
`dict(mapping)` -> new dictionary initialized from a mapping object's (key, value) pairs

`dict(iterable)` -> new dictionary initialized as if via: `d = {} for k, v in iterable:`
`d[k] = v`

`dict(**kwargs)` -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: `dict(one=1, two=2)`

5.7.5 Transports

class `riak.transports.transport.RiakTransport`

Class to encapsulate transport details and methods. All protocol transports are subclasses of this class.

__get_index_mapred_emu (*bucket, index, startkey, endkey=None*)
 Emulates a secondary index request via MapReduce. Used in the case where the transport supports MapReduce but has no native secondary index query capability.

__search_mapred_emu (*index, query*)
 Emulates a search request via MapReduce. Used in the case where the transport supports MapReduce but has no native search capability.

clear_bucket_props (*bucket*)
 Reset bucket properties to their defaults

create_search_index (*index, schema=None, n_val=None, timeout=None*)
 Creates a yokozuna search index.

create_search_schema (*schema, content*)
 Creates a yokozuna search schema.

delete (*robject, rw=None, r=None, w=None, dw=None, pr=None, pw=None, timeout=None*)
 Deletes an object.

delete_search_index (*index*)
 Deletes a yokozuna search index.

fetch_datatype (*bucket, key, r=None, pr=None, basic_quorum=None, notfound_ok=None, timeout=None, include_context=None*)
Fetches a Riak Datatype.

fulltext_add (*index, *docs*)
Adds documents to the full-text index.

fulltext_delete (*index, docs=None, queries=None*)
Removes documents from the full-text index.

get (*robject, r=None, pr=None, timeout=None, basic_quorum=None, notfound_ok=None*)
Fetches an object.

get_bucket_props (*bucket*)
Fetches properties for the given bucket.

get_bucket_type_props (*bucket_type*)
Fetches properties for the given bucket-type.

get_buckets (*bucket_type=None, timeout=None*)
Gets the list of buckets as strings.

get_client_id ()
Fetch the client id for the transport.

get_counter (*bucket, key, r=None, pr=None, basic_quorum=None, notfound_ok=None*)
Gets the value of a counter.

get_index (*bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None*)
Performs a secondary index query.

get_keys (*bucket, timeout=None*)
Lists all keys within the given bucket.

get_preflist (*bucket, key*)
Fetches the preflight for a bucket/key.

get_search_index (*index*)
Returns a yokozuna search index or None.

get_search_schema (*schema*)
Returns a yokozuna search schema.

list_search_indexes ()
Lists all yokozuna search indexes.

classmethod make_fixed_client_id ()
Returns a unique identifier for the current machine/process/thread.

classmethod make_random_client_id ()
Returns a random client identifier

mapred (*inputs, query, timeout=None*)
Sends a MapReduce request synchronously.

ping ()
Ping the remote server

put (*robject, w=None, dw=None, pw=None, return_body=None, if_none_match=None, timeout=None*)
Stores an object.

search (*index, query, **params*)
Performs a search query.

set_bucket_props (*bucket, props*)
 Sets properties on the given bucket.

set_bucket_type_props (*bucket_type, props*)
 Sets properties on the given bucket-type.

set_client_id (*client_id*)
 Set the client id. This overrides the default, random client id, which is automatically generated when none is specified in when creating the transport object.

stream_buckets (*bucket_type=None, timeout=None*)
 Streams the list of buckets through an iterator

stream_index (*bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None*)
 Streams a secondary index query.

stream_keys (*bucket, timeout=None*)
 Streams the list of keys for the bucket through an iterator.

stream_mapred (*inputs, query, timeout=None*)
 Streams the results of a MapReduce request through an iterator.

ts_delete (*table, key*)
 Deletes a timeseries object.

ts_describe (*table*)
 Retrieves a timeseries table description.

ts_get (*table, key*)
 Retrieves a timeseries object.

ts_put (*tsobj*)
 Stores a timeseries object.

ts_query (*table, query, interpolations=None*)
 Query timeseries data.

ts_stream_keys (*table, timeout=None*)
 Streams the list of keys for the table through an iterator.

update_counter (*bucket, key, value, w=None, dw=None, pw=None, returnvalue=False*)
 Updates a counter by the given value.

update_datatype (*datatype, w=None, dw=None, pw=None, return_body=None, timeout=None, include_context=None*)
 Updates a Riak Datatype by sending local operations to the server.

client_id
 the client ID for this connection

class `riak.transports.feature_detect.FeatureDetection`

Implements boolean methods that can be checked for the presence of specific server-side features. Subclasses must implement the `_server_version()` method to use this functionality, which should return the server's version as a string.

FeatureDetection is a parent class of *RiakTransport*.

_server_version ()

Gets the server version from the server. To be implemented by the individual transport class.

Return type string

bucket_stream()
Whether streaming bucket lists are supported.
Return type bool

bucket_types()
Whether bucket-types are supported.
Return type bool

client_timeouts()
Whether client-supplied timeouts are supported.
Return type bool

counters()
Whether CRDT counters are supported.
Return type bool

datatypes()
Whether datatypes are supported.
Return type bool

index_term_regex()
Whether secondary indexes supports a regexp term filter.
Return type bool

pb_all_bucket_props()
Whether all normal bucket properties are supported over Protocol Buffers.
Return type bool

pb_clear_bucket_props()
Whether bucket properties can be cleared over Protocol Buffers.
Return type bool

pb_conditionals()
Whether conditional fetch/store semantics are supported over Protocol Buffers
Return type bool

pb_head()
Whether partial-fetches (vclock and metadata only) are supported over Protocol Buffers
Return type bool

pb_indexes()
Whether secondary index queries are supported over Protocol Buffers
Return type bool

pb_search()
Whether search queries are supported over Protocol Buffers
Return type bool

pb_search_admin()
Whether search administration is supported over Protocol Buffers
Return type bool

phaseless_mapred()
Whether MapReduce requests can be submitted without phases.

Return type bool

preflists()

Whether bucket/key preflists are supported.

Return type bool

quorum_controls()

Whether additional quorums and FSM controls are available, e.g. primary quorums, basic_quorum, not-found_ok

Return type bool

stream_indexes()

Whether secondary indexes support streaming responses.

Return type bool

tombstone_vclocks()

Whether 'not found' responses might include vclocks

Return type bool

write_once()

Whether write-once operations are supported.

Return type bool

Security helpers

`riak.transports.security.verify_cb(conn, cert, errnum, depth, ok)`

The default OpenSSL certificate verification callback.

class `riak.transports.security.RiakWrappedSocket(connection, socket)`

API-compatibility wrapper for Python OpenSSL's Connection-class.

Parameters

- **connection** (`OpenSSL.SSL.Connection`) – OpenSSL connection
- **socket** (`socket`) – Underlying already connected socket

class `riak.transports.security.fileobject(sock, mode='rb', bufsize=-1, close=False)`

Extension of the socket module's fileobject to use PyOpenSSL.

`SecurityCreds._check_revoked_cert(ssl_socket)`

Checks whether the server certificate on the passed socket has been revoked by checking the CRL.

Parameters `ssl_socket` – the SSL/TLS socket

Return type bool

Raises `SecurityError` – when the certificate has been revoked

`SecurityCreds._has_credential(key)`

True if a credential or filename value has been supplied for the given property.

Parameters `key` (`str`) – which configuration property to check for

Return type bool

HTTP Transport

class `riak.transports.http.RiakHttpPool` (*client*, ***options*)

A pool of HTTP(S) transport connections.

`riak.transports.http.is_retryable` (*err*)

Determines if the given exception is something that is network/socket-related and should thus cause the HTTP connection to close and the operation retried on another node.

Return type boolean

class `riak.transports.http.RiakHttpTransport` (*node=None*, *client=None*, *connection_class=<class httplib.HTTPConnection>*, *client_id=None*, ***options*)

The RiakHttpTransport object holds information necessary to connect to Riak via HTTP.

Construct a new HTTP connection to Riak.

clear_bucket_props (*bucket*)

reset the properties on the bucket object given

create_search_index (*index*, *schema=None*, *n_val=None*, *timeout=None*)

Create a Solr search index for Yokozuna.

Parameters

- **index** (*string*) – a name of a yz index
- **schema** (*string*) – XML of Solr schema
- **n_val** (*int*) – N value of the write
- **timeout** (*integer*, *None*) – optional timeout (in ms)

:rtype boolean

create_search_schema (*schema*, *content*)

Create a new Solr schema for Yokozuna.

Parameters

- **schema** (*string*) – name of Solr schema
- **content** (*string*) – actual definition of schema (XML)

:rtype boolean

delete (*rojb*, *rw=None*, *r=None*, *w=None*, *dw=None*, *pr=None*, *pw=None*, *timeout=None*)

Delete an object.

delete_search_index (*index*)

Fetch the specified Solr search index for Yokozuna.

Parameters **index** (*string*) – a name of a yz index

:rtype boolean

fulltext_add (*index*, *docs*)

Adds documents to the search index.

fulltext_delete (*index*, *docs=None*, *queries=None*)

Removes documents from the full-text index.

get (*rojb*, *r=None*, *pr=None*, *timeout=None*, *basic_quorum=None*, *notfound_ok=None*)

Get a bucket/key from the server

get_bucket_props (*bucket*)
Get properties for a bucket

get_bucket_type_props (*bucket_type*)
Get properties for a bucket-type

get_buckets (*bucket_type=None, timeout=None*)
Fetch a list of all buckets

get_index (*bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None*)
Performs a secondary index query.

get_keys (*bucket, timeout=None*)
Fetch a list of keys for the bucket

get_preflist (*bucket, key*)
Get the preflight for a bucket/key

Parameters

- **bucket** (*RiakBucket*) – Riak Bucket
- **key** (*string*) – Riak Key

Return type list of dicts

get_resources ()
Gets a JSON mapping of server-side resource names to paths :rtype dict

get_search_index (*index*)
Fetch the specified Solr search index for Yokozuna.

Parameters **index** (*string*) – a name of a yz index
:rtype string

get_search_schema (*schema*)
Fetch a Solr schema from Yokozuna.

Parameters **schema** (*string*) – name of Solr schema
:rtype dict

list_search_indexes ()
Return a list of Solr search indexes from Yokozuna.
:rtype list of dicts

mapred (*inputs, query, timeout=None*)
Run a MapReduce query.

ping ()
Check server is alive over HTTP

put (*robject, w=None, dw=None, pw=None, return_body=True, if_none_match=False, timeout=None*)
Puts a (possibly new) object.

search (*index, query, **params*)
Performs a search query.

set_bucket_props (*bucket, props*)
Set the properties on the bucket object given

set_bucket_type_props (*bucket_type, props*)
Set the properties on the bucket-type

stats ()
 Gets performance statistics and server information

stream_buckets (*bucket_type=None, timeout=None*)
 Stream list of buckets through an iterator

stream_index (*bucket, index, startkey, endkey=None, return_terms=None, max_results=None, continuation=None, timeout=None, term_regex=None*)
 Streams a secondary index query.

Protocol Buffers Transport

class `riak.transports.pbc.RiakPbcTransport` (*node=None, client=None, timeout=None, *unused_options*)

The RiakPbcTransport object holds a connection to the protocol buffers interface on the riak server.

Construct a new RiakPbcTransport object.

clear_bucket_props (*bucket*)
 Clear bucket properties, resetting them to their defaults

get (*obj, r=None, pr=None, timeout=None, basic_quorum=None, notfound_ok=None*)
 Serialize get request and deserialize response

get_bucket_props (*bucket*)
 Serialize bucket property request and deserialize response

get_bucket_type_props (*bucket_type*)
 Fetch bucket-type properties

get_buckets (*bucket_type=None, timeout=None*)
 Serialize bucket listing request and deserialize response

get_keys (*bucket, timeout=None*)
 Lists all keys within a bucket.

get_preflist (*bucket, key*)
 Get the preflight for a bucket/key

Parameters

- **bucket** (*RiakBucket*) – Riak Bucket
- **key** (*string*) – Riak Key

Return type list of dicts

get_server_info ()
 Get information about the server

ping ()
 Ping the remote server

set_bucket_props (*bucket, props*)
 Serialize set bucket property request and deserialize response

set_bucket_type_props (*bucket_type, props*)
 Set bucket-type properties

stream_buckets (*bucket_type=None, timeout=None*)
 Stream list of buckets through an iterator

stream_keys (*bucket, timeout=None*)
 Streams keys from a bucket, returning an iterator that yields lists of keys.

ts_stream_keys (*table*, *timeout=None*)

Streams keys from a timeseries table, returning an iterator that yields lists of keys.

client_id

the client ID for this connection

5.7.6 Utilities

Link wrapper class

class `riak.mapreduce.RiakLink` (*bucket*, *key*, *tag*)

Links are just bucket/key/tag tuples, this class provides a backwards-compatible format: `RiakLink(bucket, key, tag)`

Multi-valued Dict

class `riak.multidict.MultiDict` (**args*, ***kw*)

An ordered dictionary that can have multiple values for each key. Adds the methods `getall`, `getone`, `mixed`, and `add` to the normal dictionary interface.

add (*key*, *value*)

Add the key and value, not overwriting any previous value.

getall (*key*)

Return a list of all values matching the key (may be an empty list)

getone (*key*)

Get one value matching the key, raising a `KeyError` if multiple values were found.

mixed ()

Returns a dictionary where the values are either single values, or a list of values when a key/value appears more than once in this dictionary. This is similar to the kind of dictionary often used to represent the variables in a web request.

dict_of_lists ()

Returns a dictionary where each key is associated with a list of values.

Micro-benchmarking

`riak.benchmark.measure` ()

Runs a benchmark once when used as a context manager. Example:

```
with riak.benchmark.measure() as b:
    with b.report("pow"):
        for _ in range(10000):
            math.pow(2, 10000)
    with b.report("factorial"):
        for i in range(100):
            math.factorial(i)
```

`riak.benchmark.measure_with_rehearsal` ()

Runs a benchmark when used as an iterator, injecting a garbage collection between iterations. Example:

```
for b in riak.benchmark.measure_with_rehearsal():
    with b.report("pow"):
        for _ in range(10000):
```

```

        math.pow(2,10000)
    with b.report("factorial"):
        for i in range(100):
            math.factorial(i)

```

class riak.benchmark.**Benchmark** (*rehearse=False*)

A benchmarking run, which may consist of multiple steps. See `measure_with_rehearsal()` and `measure()` for examples.

Creates a new benchmark reporter.

Parameters **rehearse** (*boolean*) – whether to run twice to take counter the effects of garbage collection

next ()

Runs the next iteration of the benchmark.

report (*name*)

Returns a report for the current step of the benchmark.

Miscellaneous

riak.util.**quacks_like_dict** (*object*)

Check if object is dict-like

riak.util.**deep_merge** (*a, b*)

Merge two deep dicts non-destructively

Uses a stack to avoid maximum recursion depth exceptions

```

>>> a = {'a': 1, 'b': {1: 1, 2: 2}, 'd': 6}
>>> b = {'c': 3, 'b': {2: 7}, 'd': {'z': [1, 2, 3]}}
>>> c = deep_merge(a, b)
>>> from pprint import pprint; pprint(c)
{'a': 1, 'b': {1: 1, 2: 7}, 'c': 3, 'd': {'z': [1, 2, 3]}}

```

riak.util.**deprecated** (*message, stacklevel=3*)

Prints a deprecation warning to the console.

class riak.util.**lazy_property** (*fget*)

A method decorator meant to be used for lazy evaluation and memoization of an object attribute. The property should represent immutable data, as it replaces itself on first access.

5.7.7 distutils commands

class commands.**build_messages** (*dist*)

Generates message code mappings. Add to the build process using:

```

setup(cmd_class={'build_messages': build_messages})

```

Create and initialize a new Command object. Most importantly, invokes the ‘initialize_options()’ method, which is the real initializer and depends on the actual command being instantiated.

finalize_options ()

initialize_options ()

run ()

description = ‘generate protocol message code mappings’

```
user_options = [('source=', None, 'source CSV file containing message code mappings'), ('destination=', None, 'destin
```

```
class commands.setup_timeseries (dist)
```

Creates bucket-types appropriate for timeseries.

Create and initialize a new Command object. Most importantly, invokes the 'initialize_options()' method, which is the real initializer and depends on the actual command being instantiated.

```
description = 'create bucket-types used in timeseries tests'
```

```
user_options = [('riak-admin=', None, 'path to the riak-admin script')]
```

Version extraction (`version` module)

Gets the current version number. If in a git repository, it is the current git tag. Otherwise it is the one contained in the PKG-INFO file.

To use this script, simply import it in your setup.py file and use the results of `get_version()` as your package version:

```
from version import *

setup(
    version=get_version()
)
```


C

commands, [70](#)

V

version, [71](#)

Symbols

- `__contains__()` (riak.datatypes.map.TypedMapView method), 60
 - `__delitem__()` (riak.datatypes.map.TypedMapView method), 61
 - `__eq__()` (riak.client.index_page.IndexPage method), 43
 - `__getitem__()` (riak.client.index_page.IndexPage method), 43
 - `__getitem__()` (riak.datatypes.map.TypedMapView method), 61
 - `__iter__()` (riak.client.index_page.IndexPage method), 43
 - `__iter__()` (riak.datatypes.map.TypedMapView method), 61
 - `__len__()` (riak.datatypes.map.TypedMapView method), 61
 - `_acquire()` (riak.client.transport.RiakClientTransport method), 58
 - `_check_revoked_cert()` (riak.security.SecurityCreds method), 65
 - `_check_type()` (riak.datatypes.Datatype method), 60
 - `_choose_pool()` (riak.client.transport.RiakClientTransport method), 58
 - `_coerce_value()` (riak.datatypes.Datatype method), 60
 - `_default_value()` (riak.datatypes.Datatype method), 60
 - `_fetcher()` (riak.client.multiget.MultiGetPool method), 59
 - `_get_index_mapred_emu()` (riak.transports.transport.RiakTransport method), 61
 - `_has_credential()` (riak.security.SecurityCreds method), 65
 - `_is_retryable()` (in module riak.client.transport), 59
 - `_post_init()` (riak.datatypes.Datatype method), 60
 - `_require_context()` (riak.datatypes.Datatype method), 60
 - `_search_mapred_emu()` (riak.transports.transport.RiakTransport method), 61
 - `_server_version()` (riak.transports.feature_detect.FeatureDetection method), 63
 - `_should_quit()` (riak.client.multiget.MultiGetPool method), 59
 - `_transport()` (riak.client.transport.RiakClientTransport method), 58
 - `_type_error_msg` (riak.datatypes.Datatype attribute), 60
 - `_with_retries()` (riak.client.transport.RiakClientTransport method), 58
- ## A
- `acquire()` (riak.transports.pool.Pool method), 57
 - `add()` (riak.datatypes.Set method), 40
 - `add()` (riak.mapreduce.RiakMapReduce method), 44
 - `add()` (riak.multidict.MultiDict method), 69
 - `add()` (riak.riak_object.RiakObject method), 49
 - `add_bucket()` (riak.mapreduce.RiakMapReduce method), 44
 - `add_bucket_key_data()` (riak.mapreduce.RiakMapReduce method), 45
 - `add_index()` (riak.riak_object.RiakObject method), 35
 - `add_key_filter()` (riak.mapreduce.RiakMapReduce method), 44
 - `add_key_filters()` (riak.mapreduce.RiakMapReduce method), 44
 - `add_link()` (riak.riak_object.RiakObject method), 36
 - `add_object()` (riak.mapreduce.RiakMapReduce method), 45
 - `allow_mult` (riak.bucket.RiakBucket attribute), 27
 - `assign()` (riak.datatypes.Register method), 41
- ## B
- `BadResource`, 56
 - `Benchmark` (class in riak.benchmark), 70
 - `bucket` (riak.riak_object.RiakObject attribute), 33
 - `bucket()` (riak.bucket.BucketType method), 30
 - `bucket()` (riak.client.RiakClient method), 12
 - `bucket_stream()` (riak.transports.feature_detect.FeatureDetection method), 63
 - `bucket_type` (riak.bucket.RiakBucket attribute), 26
 - `bucket_type()` (riak.client.RiakClient method), 12
 - `bucket_types()` (riak.transports.feature_detect.FeatureDetection method), 64
 - `BucketType` (class in riak.bucket), 30
 - `build_messages` (class in commands), 70

C

cacert (riak.security.SecurityCreds attribute), 56
 cert (riak.security.SecurityCreds attribute), 56
 charset (riak.riak_object.RiakObject attribute), 35
 ciphers (riak.security.SecurityCreds attribute), 56
 claimed (riak.transports.pool.Resource attribute), 56
 clear() (riak.datatypes.Datatype method), 39
 clear() (riak.transports.pool.Pool method), 57
 clear_bucket_props() (riak.client.RiakClient method), 13
 clear_bucket_props() (riak.transports.http.RiakHttpTransport method), 66
 clear_bucket_props() (riak.transports.pbc.RiakPbcTransport method), 68
 clear_bucket_props() (riak.transports.transport.RiakTransport method), 61
 clear_properties() (riak.bucket.RiakBucket method), 26
 client_id (riak.client.RiakClient attribute), 10
 client_id (riak.transports.pbc.RiakPbcTransport attribute), 69
 client_id (riak.transports.transport.RiakTransport attribute), 63
 client_timeouts() (riak.transports.feature_detect.FeatureDetection method), 64
 commands (module), 70
 ConflictError, 37
 content_encoding (riak.riak_object.RiakObject attribute), 35
 content_type (riak.riak_object.RiakObject attribute), 35
 context (riak.datatypes.Datatype attribute), 38
 continuation (riak.client.index_page.IndexPage attribute), 43
 Counter (class in riak.datatypes), 39
 counters (riak.datatypes.Map attribute), 40
 counters() (riak.transports.feature_detect.FeatureDetection method), 64
 create_resource() (riak.transports.pool.Pool method), 57
 create_search_index() (riak.client.RiakClient method), 22
 create_search_index() (riak.transports.http.RiakHttpTransport method), 66
 create_search_index() (riak.transports.transport.RiakTransport method), 61
 create_search_schema() (riak.client.RiakClient method), 22
 create_search_schema() (riak.transports.http.RiakHttpTransport method), 66
 create_search_schema() (riak.transports.transport.RiakTransport method), 61
 crl (riak.security.SecurityCreds attribute), 56

D

data (riak.riak_object.RiakObject attribute), 35
 Datatype (class in riak.datatypes), 38
 datatype (riak.bucket.BucketType attribute), 31
 Datatype.store() (in module riak.datatypes), 39

datatypes() (riak.transports.feature_detect.FeatureDetection method), 64
 decrement() (riak.datatypes.Counter method), 39
 deep_merge() (in module riak.util), 70
 default_encoder() (in module riak.client), 22
 default_resolver() (in module riak.resolver), 36
 DEFAULT_RETRY_COUNT (in module riak.client.transport), 11
 delete() (riak.bucket.RiakBucket method), 28
 delete() (riak.client.RiakClient method), 15
 delete() (riak.datatypes.Datatype method), 39
 delete() (riak.riak_object.RiakObject method), 34
 delete() (riak.transports.http.RiakHttpTransport method), 66
 delete() (riak.transports.transport.RiakTransport method), 61
 delete_resource() (riak.transports.pool.Pool method), 57
 delete_search_index() (riak.client.RiakClient method), 22
 delete_search_index() (riak.transports.http.RiakHttpTransport method), 66
 delete_search_index() (riak.transports.transport.RiakTransport method), 61
 deprecated() (in module riak.util), 70
 description (commands.build_messages attribute), 70
 description (commands.setup_timeseries attribute), 71
 destroy_resource() (riak.transports.pool.Pool method), 57
 dict_of_lists() (riak.multidict.MultiDict method), 69
 disable() (riak.datatypes.Flag method), 42
 disable_search() (riak.bucket.RiakBucket method), 32
 discard() (riak.datatypes.Set method), 40
 dw (riak.bucket.RiakBucket attribute), 27

E

enable() (riak.datatypes.Flag method), 42
 enable_search() (riak.bucket.RiakBucket method), 32
 encoded_data (riak.riak_object.RiakObject attribute), 35
 enq() (riak.client.multiget.MultiGetPool method), 59
 etag (riak.riak_object.RiakObject attribute), 35
 exists (riak.riak_object.RiakObject attribute), 33

F

FeatureDetection (class in riak.transports.feature_detect), 63
 fetch_datatype() (riak.client.RiakClient method), 15
 fetch_datatype() (riak.transports.transport.RiakTransport method), 61
 fileobject (class in riak.transports.security), 65
 filter_not_found() (riak.mapreduce.RiakMapReduce method), 48
 finalize_options() (commands.build_messages method), 70
 Flag (class in riak.datatypes), 41
 flags (riak.datatypes.Map attribute), 41
 fulltext_add() (riak.client.RiakClient method), 23

fulltext_add() (riak.transports.http.RiakHttpTransport method), 66
 fulltext_add() (riak.transports.transport.RiakTransport method), 62
 fulltext_delete() (riak.client.RiakClient method), 24
 fulltext_delete() (riak.transports.http.RiakHttpTransport method), 66
 fulltext_delete() (riak.transports.transport.RiakTransport method), 62
 fulltext_search() (riak.client.RiakClient method), 20

G

get() (riak.bucket.RiakBucket method), 28
 get() (riak.client.RiakClient method), 14
 get() (riak.transports.http.RiakHttpTransport method), 66
 get() (riak.transports.pbc.RiakPbcTransport method), 68
 get() (riak.transports.transport.RiakTransport method), 62
 get_bucket_props() (riak.client.RiakClient method), 13
 get_bucket_props() (riak.transports.http.RiakHttpTransport method), 66
 get_bucket_props() (riak.transports.pbc.RiakPbcTransport method), 68
 get_bucket_props() (riak.transports.transport.RiakTransport method), 62
 get_bucket_type_props() (riak.client.RiakClient method), 12
 get_bucket_type_props() (riak.transports.http.RiakHttpTransport method), 67
 get_bucket_type_props() (riak.transports.pbc.RiakPbcTransport method), 68
 get_bucket_type_props() (riak.transports.transport.RiakTransport method), 62
 get_buckets() (riak.bucket.BucketType method), 31
 get_buckets() (riak.client.RiakClient method), 11
 get_buckets() (riak.transports.http.RiakHttpTransport method), 67
 get_buckets() (riak.transports.pbc.RiakPbcTransport method), 68
 get_buckets() (riak.transports.transport.RiakTransport method), 62
 get_client_id() (riak.transports.transport.RiakTransport method), 62
 get_counter() (riak.bucket.RiakBucket method), 32
 get_counter() (riak.client.RiakClient method), 24
 get_counter() (riak.transports.transport.RiakTransport method), 62
 get_decoder() (riak.bucket.RiakBucket method), 29
 get_decoder() (riak.client.RiakClient method), 23
 get_encoder() (riak.bucket.RiakBucket method), 29
 get_encoder() (riak.client.RiakClient method), 22
 get_index() (riak.bucket.RiakBucket method), 29

get_index() (riak.client.RiakClient method), 19
 get_index() (riak.transports.http.RiakHttpTransport method), 67
 get_index() (riak.transports.transport.RiakTransport method), 62
 get_keys() (riak.bucket.RiakBucket method), 30
 get_keys() (riak.client.RiakClient method), 13
 get_keys() (riak.transports.http.RiakHttpTransport method), 67
 get_keys() (riak.transports.pbc.RiakPbcTransport method), 68
 get_keys() (riak.transports.transport.RiakTransport method), 62
 get_preflist() (riak.transports.http.RiakHttpTransport method), 67
 get_preflist() (riak.transports.pbc.RiakPbcTransport method), 68
 get_preflist() (riak.transports.transport.RiakTransport method), 62
 get_properties() (riak.bucket.BucketType method), 31
 get_properties() (riak.bucket.RiakBucket method), 26
 get_property() (riak.bucket.BucketType method), 31
 get_property() (riak.bucket.RiakBucket method), 26
 get_resources() (riak.transports.http.RiakHttpTransport method), 67
 get_search_index() (riak.client.RiakClient method), 22
 get_search_index() (riak.transports.http.RiakHttpTransport method), 67
 get_search_index() (riak.transports.transport.RiakTransport method), 62
 get_search_schema() (riak.client.RiakClient method), 22
 get_search_schema() (riak.transports.http.RiakHttpTransport method), 67
 get_search_schema() (riak.transports.transport.RiakTransport method), 62
 get_server_info() (riak.transports.pbc.RiakPbcTransport method), 68
 getall() (riak.multidict.MultiDict method), 69
 getone() (riak.multidict.MultiDict method), 69

H

has_next_page() (riak.client.index_page.IndexPage method), 43

I

increment() (riak.datatypes.Counter method), 39
 index() (riak.mapreduce.RiakMapReduce method), 45
 index_term_regex() (riak.transports.feature_detect.FeatureDetection method), 64
 indexes (riak.riak_object.RiakObject attribute), 35
 IndexPage (class in riak.client.index_page), 43
 initialize_options() (commands.build_messages method), 70
 is_default() (riak.bucket.BucketType method), 30

is_retryable() (in module riak.transports.http), 66

K

key (riak.riak_object.RiakObject attribute), 33

L

last_modified (riak.riak_object.RiakObject attribute), 35

last_written_resolver() (in module riak.resolver), 36

lazy_property (class in riak.util), 70

link() (riak.mapreduce.RiakMapReduce method), 46

link() (riak.riak_object.RiakObject method), 49

links (riak.riak_object.RiakObject attribute), 35

list_search_indexes() (riak.client.RiakClient method), 22

list_search_indexes() (riak.transports.http.RiakHttpTransport method), 67

list_search_indexes() (riak.transports.transport.RiakTransport method), 62

M

make_fixed_client_id() (riak.transports.transport.RiakTransport class method), 62

make_random_client_id()
(riak.transports.transport.RiakTransport class method), 62

Map (class in riak.datatypes), 40

map() (riak.mapreduce.RiakMapReduce method), 46

map() (riak.riak_object.RiakObject method), 49

map_values() (riak.mapreduce.RiakMapReduce method), 47

map_values_json() (riak.mapreduce.RiakMapReduce method), 47

mapred() (riak.client.RiakClient method), 18

mapred() (riak.transports.http.RiakHttpTransport method), 67

mapred() (riak.transports.transport.RiakTransport method), 62

maps (riak.datatypes.Map attribute), 41

measure() (in module riak.benchmark), 69

measure_with_rehearsal() (in module riak.benchmark), 69

mixed() (riak.multidict.MultiDict method), 69

modified (riak.datatypes.Datatype attribute), 38

MultiDict (class in riak.multidict), 69

multiget() (in module riak.client.multiget), 60

multiget() (riak.bucket.RiakBucket method), 28

multiget() (riak.client.RiakClient method), 15

MultiGetPool (class in riak.client.multiget), 59

N

n_val (riak.bucket.RiakBucket attribute), 26

name (riak.bucket.BucketType attribute), 30

name (riak.bucket.RiakBucket attribute), 26

new() (riak.bucket.RiakBucket method), 27

new_from_file() (riak.bucket.RiakBucket method), 27

next() (riak.benchmark.Benchmark method), 70

next_page() (riak.client.index_page.IndexPage method), 43

nodes (riak.client.RiakClient attribute), 10

O

object (riak.transports.pool.Resource attribute), 57

P

paginate_index() (riak.bucket.RiakBucket method), 29

paginate_index() (riak.client.RiakClient method), 20

paginate_stream_index() (riak.bucket.RiakBucket method), 29

paginate_stream_index() (riak.client.RiakClient method), 21

password (riak.security.SecurityCreds attribute), 56

pb_all_bucket_props() (riak.transports.feature_detect.FeatureDetection method), 64

pb_clear_bucket_props()
(riak.transports.feature_detect.FeatureDetection method), 64

pb_conditionals() (riak.transports.feature_detect.FeatureDetection method), 64

pb_head() (riak.transports.feature_detect.FeatureDetection method), 64

pb_indexes() (riak.transports.feature_detect.FeatureDetection method), 64

pb_search() (riak.transports.feature_detect.FeatureDetection method), 64

pb_search_admin() (riak.transports.feature_detect.FeatureDetection method), 64

phaseless_mapred() (riak.transports.feature_detect.FeatureDetection method), 64

ping() (riak.client.RiakClient method), 11

ping() (riak.transports.http.RiakHttpTransport method), 67

ping() (riak.transports.pbc.RiakPbcTransport method), 68

ping() (riak.transports.transport.RiakTransport method), 62

pkey (riak.security.SecurityCreds attribute), 56

Pool (class in riak.transports.pool), 57

pool (riak.transports.pool.Resource attribute), 57

POOL_SIZE (in module riak.client.multiget), 59

PoolIterator (class in riak.transports.pool), 58

pr (riak.bucket.RiakBucket attribute), 27

preflists() (riak.transports.feature_detect.FeatureDetection method), 65

protocol (riak.client.RiakClient attribute), 10

PROTOCOLS (riak.client.RiakClient attribute), 10

put() (riak.client.RiakClient method), 15

put() (riak.transports.http.RiakHttpTransport method), 67

put() (riak.transports.transport.RiakTransport method), 62

pw (riak.bucket.RiakBucket attribute), 27

Q

quacks_like_dict() (in module riak.util), 70

quorum_controls() (riak.transports.feature_detect.FeatureDetection method), 65

R

r (riak.bucket.RiakBucket attribute), 27

reduce() (riak.mapreduce.RiakMapReduce method), 46

reduce() (riak.riak_object.RiakObject method), 49

reduce_limit() (riak.mapreduce.RiakMapReduce method), 48

reduce_max() (riak.mapreduce.RiakMapReduce method), 47

reduce_min() (riak.mapreduce.RiakMapReduce method), 47

reduce_numeric_sort() (riak.mapreduce.RiakMapReduce method), 48

reduce_slice() (riak.mapreduce.RiakMapReduce method), 48

reduce_sort() (riak.mapreduce.RiakMapReduce method), 47

reduce_sum() (riak.mapreduce.RiakMapReduce method), 47

Register (class in riak.datatypes), 41

registers (riak.datatypes.Map attribute), 41

release() (riak.transports.pool.Pool method), 58

release() (riak.transports.pool.Resource method), 56

reload() (riak.datatypes.Datatype method), 38

reload() (riak.riak_object.RiakObject method), 34

remove_index() (riak.riak_object.RiakObject method), 35

report() (riak.benchmark.Benchmark method), 70

resolver (riak.bucket.RiakBucket attribute), 26

resolver (riak.client.RiakClient attribute), 10

resolver (riak.riak_object.RiakObject attribute), 33

Resource (class in riak.transports.pool), 56

retries (riak.client.RiakClient attribute), 10

retries (riak.client.transport.RiakClientTransport attribute), 59

retry_count() (riak.client.RiakClient method), 10

retry_count() (riak.client.transport.RiakClientTransport method), 58

retryable() (in module riak.client.transport), 59

retryableHttpOnly() (in module riak.client.transport), 59

RIAK_MULTIGET_POOL (in module riak.client.multiget), 59

RiakBucket (class in riak.bucket), 26

RiakClient (class in riak.client), 9

RiakClientTransport (class in riak.client.transport), 58

RiakContent (class in riak.content), 36

RiakHttpPool (class in riak.transports.http), 66

RiakHttpTransport (class in riak.transports.http), 66

RiakKeyFilter (class in riak.mapreduce), 45

RiakLink (class in riak.mapreduce), 69

RiakLinkPhase (class in riak.mapreduce), 47

RiakMapReduce (class in riak.mapreduce), 44

RiakMapReducePhase (class in riak.mapreduce), 46

RiakNode (class in riak.node), 10

RiakObject (class in riak.riak_object), 33

RiakPbcTransport (class in riak.transports.pbc), 68

RiakTransport (class in riak.transports.transport), 61

RiakWrappedSocket (class in riak.transports.security), 65

run() (commands.build_messages method), 70

run() (riak.mapreduce.RiakMapReduce method), 48

rw (riak.bucket.RiakBucket attribute), 27

S

search() (riak.bucket.RiakBucket method), 29

search() (riak.mapreduce.RiakMapReduce method), 45

search() (riak.transports.http.RiakHttpTransport method), 67

search() (riak.transports.transport.RiakTransport method), 62

search_enabled() (riak.bucket.RiakBucket method), 32

SecurityCreds (class in riak.security), 55

Set (class in riak.datatypes), 40

set_bucket_props() (riak.client.RiakClient method), 13

set_bucket_props() (riak.transports.http.RiakHttpTransport method), 67

set_bucket_props() (riak.transports.pbc.RiakPbcTransport method), 68

set_bucket_props() (riak.transports.transport.RiakTransport method), 62

set_bucket_type_props() (riak.client.RiakClient method), 12

set_bucket_type_props() (riak.transports.http.RiakHttpTransport method), 67

set_bucket_type_props() (riak.transports.pbc.RiakPbcTransport method), 68

set_bucket_type_props() (riak.transports.transport.RiakTransport method), 63

set_client_id() (riak.transports.transport.RiakTransport method), 63

set_decoder() (riak.bucket.RiakBucket method), 30

set_decoder() (riak.client.RiakClient method), 23

set_encoder() (riak.bucket.RiakBucket method), 29

set_encoder() (riak.client.RiakClient method), 23

set_index() (riak.riak_object.RiakObject method), 35

set_properties() (riak.bucket.BucketType method), 31

set_properties() (riak.bucket.RiakBucket method), 26

set_property() (riak.bucket.BucketType method), 31

set_property() (riak.bucket.RiakBucket method), 26

sets (riak.datatypes.Map attribute), 41

setup_timeseries (class in commands), 71

siblings (riak.riak_object.RiakObject attribute), 36

ssl_version (riak.security.SecurityCreds attribute), 56

start() (riak.client.multiget.MultiGetPool method), 59

stats() (riak.transports.http.RiakHttpTransport method), 67

stop() (riak.client.multiget.MultiGetPool method), 59

stopped() (riak.client.multiget.MultiGetPool method), 59

store() (riak.riak_object.RiakObject method), 33

stream() (riak.mapreduce.RiakMapReduce method), 48

stream_buckets() (riak.bucket.BucketType method), 31

stream_buckets() (riak.client.RiakClient method), 11

stream_buckets() (riak.transports.http.RiakHttpTransport method), 68

stream_buckets() (riak.transports.pbc.RiakPbcTransport method), 68

stream_buckets() (riak.transports.transport.RiakTransport method), 63

stream_index() (riak.bucket.RiakBucket method), 29

stream_index() (riak.client.RiakClient method), 19

stream_index() (riak.transports.http.RiakHttpTransport method), 68

stream_index() (riak.transports.transport.RiakTransport method), 63

stream_indexes() (riak.transports.feature_detect.FeatureDetection method), 65

stream_keys() (riak.bucket.RiakBucket method), 30

stream_keys() (riak.client.RiakClient method), 14

stream_keys() (riak.transports.pbc.RiakPbcTransport method), 68

stream_keys() (riak.transports.transport.RiakTransport method), 63

stream_mapred() (riak.client.RiakClient method), 18

stream_mapred() (riak.transports.transport.RiakTransport method), 63

T

Task (class in riak.client.multiget), 59

to_op() (riak.datatypes.Datatype method), 60

tombstone_vclocks() (riak.transports.feature_detect.FeatureDetection method), 65

transaction() (riak.transports.pool.Pool method), 58

ts_delete() (riak.client.RiakClient method), 17

ts_delete() (riak.transports.transport.RiakTransport method), 63

ts_describe() (riak.client.RiakClient method), 16

ts_describe() (riak.transports.transport.RiakTransport method), 63

ts_get() (riak.client.RiakClient method), 16

ts_get() (riak.transports.transport.RiakTransport method), 63

ts_put() (riak.client.RiakClient method), 17

ts_put() (riak.transports.transport.RiakTransport method), 63

ts_query() (riak.client.RiakClient method), 17

ts_query() (riak.transports.transport.RiakTransport method), 63

ts_stream_keys() (riak.client.RiakClient method), 17

ts_stream_keys() (riak.transports.pbc.RiakPbcTransport method), 69

ts_stream_keys() (riak.transports.transport.RiakTransport method), 63

type_name (riak.datatypes.Datatype attribute), 60

TypedMapView (class in riak.datatypes.map), 60

TYPES (in module riak.datatypes), 61

U

update() (riak.datatypes.Datatype method), 39

update_counter() (riak.bucket.RiakBucket method), 32

update_counter() (riak.client.RiakClient method), 24

update_counter() (riak.transports.transport.RiakTransport method), 63

update_datatype() (riak.client.RiakClient method), 16

update_datatype() (riak.transports.transport.RiakTransport method), 63

user_options (commands.build_messages attribute), 71

user_options (commands.setup_timeseries attribute), 71

usermeta (riak.riak_object.RiakObject attribute), 35

username (riak.security.SecurityCreds attribute), 55

V

value (riak.datatypes.Counter attribute), 39

value (riak.datatypes.Datatype attribute), 38

value (riak.datatypes.Flag attribute), 41

value (riak.datatypes.Map attribute), 40

value (riak.datatypes.Register attribute), 41

value (riak.datatypes.Set attribute), 40

VClock (class in riak.riak_object), 33

vclock (riak.riak_object.RiakObject attribute), 33

verify_cb() (in module riak.transports.security), 65

version (module), 71

W

w (riak.bucket.RiakBucket attribute), 27

write_once() (riak.transports.feature_detect.FeatureDetection method), 65