Encore Documentation

Release 0.7.0

Enthought, Inc.

January 19, 2017

Contents

This package consists of a collection of core utility packages useful for building Python applications. This package is intended to be at the bottom of the software stack and have zero required external dependencies aside from the Python Standard Library.

CHAPTER 1

Packages

Events: A package implementing a lightweight application-wide Event dispatch system. Listeners can subscribe to events based on Event type or by filtering on event attributes. Typical uses include UI components listening to low-level progress notifications and change notification for distributed resources.

Storage: Abstract base classes and concrete implementations of a basic key-value storage API. The API is intended to be general purpose enough to support a variety of local and remote storage systems.

Concurrent: A package of tools for handling concurrency within applications.

Terminal: Some utilities for working with text-based terminal displays.

CHAPTER 2

Prerequisites

- Python >= 2.7 or Python >= 3.4
- Sphinx, graphviz, pydot (documentation build)
- Some optional modules have dependencies on:
  - Requests (http://docs.python-requests.org/en/latest/)
  - Futures (https://code.google.com/p/pythonfutures/)

CHAPTER 3

Contents

3.1 Events

The `encore.events` module provides a unified event system for application-level events. This is intended to be distinct from the event systems provided by UI toolkits or Traits, although there is nothing stopping an implementation from using such for a back-end.

3.1.1 Contents

Usage

The `encore.events` package provides a fairly straightforward event dispatcher.

Event Classes

Basic filtering is based upon the class of the event, so most users will want to define their own set of event classes, but a number of standard `BaseEvent` subclasses are provided by the module. An event class can be as simple as:

```
from encore.events.api import BaseEvent

class SaveEvent(BaseEvent):
    """ Event generated when a file is saved

    Attributes
    ----------
    directory : path
        The directory that the file that was saved in.
    filename : string
        The name of the file that was saved.

    """
```

The `BaseEvent`'s `__init__()` method takes any additional keyword arguments it is supplied, and adds them as attributes on the object. Every event has a source object which is the object which generated the event. You can create an instance of an event like so:

```
event = SaveEvent(source=obj, directory='foo/bar', filename='baz.txt')
```

Because filtering of events respects the class heirarchy of events, you will frequently want to define some abstract base classes to assist with filtering:

```
from encore.events.api import BaseEvent

class FileEvent(BaseEvent):
    """ Event generated when a file is operated upon

    Attributes
    ----------
    directory : path
        The directory that the file that was saved in.
    filename : string
        The name of the file that was saved.

    """

class OpenEvent(FileEvent):
    pass

class SaveEvent(FileEvent):
    pass

class DeleteEvent(FileEvent):
    pass
```

In the above example, you will probably never generate an instance of `FileEvent`, but you may set up listeners for such events.

Event Managers

To emit events, you will then need to ensure that your application has a (usually unique) event manager to handle the dispatch of events. Creating an event manager is straightforward:

```
from encore.events.api import EventManager

event_manager = EventManager()
```

More typically you will have some sort of global application state object that is responsible for managing things like event managers, and then you might use it as follows:

```
import os
from uuid import uuid4
from encore.events.api import EventManager, ProgressManager
from .app_events import SaveEvent

class App(object):
    def __init__(self):
        self.event_manager = EventManager()

class File(object):
    def __init__(self, app, directory, filename, data=''):
        self.app = app
        self.directory = directory
        self.filename = filename
        self.data = data

    def save(self):
        event_manager = self.app.event_manager
        path = os.path.join(self.directory, self.filename)
```

```
        op_id = uuid4()
        try:
            with open(path, 'wb') as fp:
                steps = range(0, len(data), 2**20)
                with ProgressManager(event_manager, self, op_id,
                        'Saving "%s"' % path, len(steps)) as progress:
                    for i, pos in enumerate(step):
                        fp.write(self.data[i:i+2**20])
                        progress('Saving "%s" (%d of %d bytes)' % (path, pos, len(data)),
                            step=i+1)
        else:
            event_manager.emit(SaveEvent(source=self, directory=self.directory,
                filename=self.filename))
```

Notice the use of the standard `ProgressManager` subclasses to generate progress update events while writing out the data.

Listeners

A listener is simply a function which expects to be given an event instance and does something with it. For example, we could write a listener which listens for `SaveEvents` and logs them to a logger:

```
import logging
import os

logger = logging.getLogger(__name__)

def save_logger(event):
    path = os.path.join(event.directory, event.filename)
    logger.info("Saved file '%s'" % path)
```

Once you have a listener, it can be connected to listen for particular classes of events via the event manager:

```
event_manager.connect(SaveEvent, save_logger)
```

Once the listener is connected, the `save_logger()` function will be called every time that a `SaveEvent` is emitted. A listener can be explicitly disconnected by calling the `disconnect()` method of the event manager:

```
event_manager.disconnect(SaveEvent, save_logger)
```

A listener which is a bound method will be disconnected automatically if the underlying instance has been garbage-collected, so in many instances you will not need to worry about explicitly disconnecting listeners.

In the above example, you would be more likely to want to log all `FileEvents` rather than save events. This could be achieved by something like:

```
def file_event_logger(event):
    path = os.path.join(event.directory, event.filename)
    logger.info("%s: file '%s'" % (event.__class__.__name__, path))

event_manager.connect(FileEvent, file_event_logger)
```

This will call the `file_event_logger()` function every time that a subclass of `FileEvent` is emitted.

Listener Priority

It is possible to have multiple listeners on a particular class, and you may want some listeners to run before other listeners. In particular, a listener may mark an event as "handled" in which case processing stops and all lower priority listeners do not get to see the event.

For instance, in the above example, we might want to have both the `save_logger()` and `file_event_logger()` active. In that case we don't want to have save events logged twice, so we can do the following:

```
def save_logger(event):
    path = os.path.join(event.directory, event.filename)
    logger.info("Saved file '%s'" % path)
    event.mark_as_handled()

event_manager.connect(SaveEvent, save_logger, priority=100)
event_manager.connect(FileEvent, file_event_logger, piority=50)
```

By setting the priority of `save_logger()` higher than that of `file_event_logger()`, it will get called first, and when it calls the event's *`mark_as_handled()`* method then it will prevent any lower-priority events from firing.

In the default event manager implementation, listeners of the same priority are called in the order in which they were connected.

Filtering

On occassion a listener may only care about events from certain sources or matching certain attributes. The event manager allows a filter to be specified when connecting a listener, so that the listener will only be called when the filter is matched.

A filter is simply specified as a dictionary of event attribute, value pairs:

```
class Project(object):
    def __init__(self, app, directory):
        self.app = app
        self.directory = directory
        self._needs_compile = False
        self._connect_listener()

    def directory_listener(self, event):
        self._needs_complie = True

    def _connect_listener(self):
        self.app.event_manager.connect(SaveEvent, self.directory_listener,
            filter={'directory': self.directory})
```

In this example, a `Project` instance will have its `directory_watcher()` method called whenever a file is saved in the directory specified by its `directory` attribute.

Example: Progress Bar

As an example which ties together the concepts which have been shown so far, we will write some code which displays progress indications to standard out that look something like the following:

```
Saving "foo/bar/baz.txt":
[**********************************
```

We start with a class which is responsible for listening for the start of a progress event. For simplicty we will assume that there will only be one progress sequence happening at any given time, so we will have the class instance hook up a listener for `ProgressStartEvents`:

```
class ProgressDisplay(object):
    def __init__(self, event_manager):
        self.event_manager = event_manager
        self.event_manager.connect(ProgressStartEvent, self.start_listener)
```

When a `ProgressStartEvent` occurs, then we will print out the initial text, and set up a listener for the `ProgressStepEvent` and `ProgressEndEvent` event types:

```
def start_listener(self, event):
    # display initial text
    sys.stdout.write(event.message)
    sys.stdout.write(':\n[')
    sys.stdout.flush()

    # create a ProgressWriter instance
    writer = ProgressWriter(self, event.operation_id, event.steps)
    self.writers[event.operation_id] = writer

    # connect listeners
    self.event_manager.connect(ProgressStepEvent, writer.step_listener,
        filter={'operation_id': event.operation_id})
    self.event_manager.connect(ProgressEndEvent, writer.end_listener,
        filter={'operation_id': event.operation_id})
```

The writer class handles listening for step and end events. The end event listener simply removes the writer object from the display, which will cause it to eventually be garbage-collected and the listeners disconnected automatically:

```
class ProgressWriter(object):
    def __init__(self, display, operation_id, steps):
        self.display = display
        self.operation_id = operation_id
        self.steps = steps
        self._count = 0
        self._max = 75

    def step_listener(self, event):
        stars = int(round(float(event.step)/self.steps*self._max))
        if stars > self._count:
            sys.stdout.write('*'*(stars-self._count))
            sys.stdout.flush()
            self._count = stars

    def end_listener(self, event):
        if event.exit_state == 'normal':
            sys.stdout.write(']\n')
            sys.stdout.flush()
        else:
            sys.stdout.write('\n')
            sys.stdout.write(event.exit_state.upper())
            sys.stdout.write(': ')
            sys.stdout.write(event.message)
            sys.stdout.write('\n')
```

```
            sys.stdout.flush()
        del self.display[self.operation_id]
```

Advanced Features

Disabling Events The event manager has methods that allow code to temporarily disable events of a certain class. These are accessed via the *disable()*, *enable()*, and *is_enabled()* methods. Disabling an event class will also disable any of its subclasses, so:

```
event_manager.disable(BaseEvent)
```

will disable all events.

Enabled/disabled state is kept track of on a per-class basis, so after:

```
event_manager.disable(SaveEvent)
event_manager.disable(FileEvent)
event_manager.enable(FileEvent)
```

the `SaveEvent` events will still be disabled.

Pre- and Post-Emit Callbacks The event classes also have two hooks *pre_emit()* and *post_emit()* which get called immediately before and immediately after dispatch to listeners. This potentially allows Event code to perform actions based upon interactions with listeners, such as having a *post_emit()* method which does something sensible if an event is not handled. These hooks may also be of use for instrumenting and debugging code.

Threading By default events are processed on the thread that they were emitted on, and the *connect()*, *disconnect()* and *emit()* methods should be thread-safe. Processing an event blocks that thread from further work until all listeners have been called.

The *emit()* method has an optional argument `block` which if `False` will cause the emit method to create a worker thread to perform the listener dispatch, and will return that thread from the function call.

Abstract Event Manager API

This module defines event manager class API.

The main class of the module is the *BaseEventManager*. Event managers are expected to implement the interface as specified by *BaseEventManager*. A concrete implementation is present in the *event_manager* module.

class `encore.events.abstract_event_manager.`**BaseEventManager**

This abstract class defines the API for Event Managers.

connect(*cls*, *func*, *filter=None*, *priority=0*)

Add a listener for the event.

Parameters

- **cls** (*class*) – The class of events for which the listener is registered.
- **func** (*callable*) – A callable to be called when the event is emitted. The function should expect one argument which is the event instance which was emitted.
- **filter** (*dict*) – Filters to match for before calling the listener. The listener is called only when the event matches all of the filter .

Filter specification:

  - **key: string which is extended (. separated) name of an** attribute of the event instance.
  - value: the value of the specified attribute.

  If the attribute does not exist then the filter is considered failed and the listener is not called.

- **priority** (*int*) – The priority of the listener. Higher priority listeners are called before lower priority listeners.

Note

The filtering is added so that future optimizations can be done on specific events with large number of handlers. For example there should be a fast way to filter key events to specific listeners rather than iterating through all listeners.

disable(*cls*)
Disable the event from generating notifications.

Parameters **cls** (*class*) – The class of events which we want to disable.

disconnect(*cls*, *func*)
Disconnects a listener from being notified about the event'

Parameters

- **cls** (*class*) – The class of events for which the listener is registered.
- **func** (*callable*) – The callable which was registered for that class.

Raises **KeyError** - if *func* is not already connected.

emit(*event*, *block=True*)
Notifies all listeners about the event with the specified arguments.

Parameters

- **event** (instance of *BaseEvent*) – The *BaseEvent* instance to emit.
- **block** (*bool*) – Whether to block the call until the event handling is finished. If block is False, the event will be emitted in a separate thread and the thread will be returned, so you can later query its status or do `wait()` on the thread.

Note

Listeners of superclasses of the event are also called. Eg. a *BaseEvent* listener will also be notified about any derived class events.

enable(*cls*)
Enable the event again to generate notifications.

Parameters **cls** (*class*) – The class of events which we want to enable.

is_enabled(*cls*)
Check if the event is enabled.

Parameters **cls** (*class*) – The class of events which we want check the status of.

Events

The module also provides the base class for all event objects.

class encore.events.abstract_event_manager.**BaseEvent**(*source=None*, ***kwargs*)
Base class for all events.

Parameters

- **source** (*object*) – The object which generated the Event.
- **kwargs** (*dict*) – Additional Event attributes which will be added to the Event object.

mark_as_handled()
Mark the event as handled so subsequent listeners are not notified.

post_emit()
Called after emitting an event.

Can be used any event specific functionality, post event validation etc.

pre_emit()
Called before emitting an event.

Can be used any event specific functionality, validation etc.

Event Manager Implementation

This module defines an event registry, notification and filtering class.

The main class of the module is the *EventManager*.

class encore.events.event_manager.**EventManager**
A single registry point for all application events.

connect(*cls*, *func*, *filter=None*, *priority=0*)
Add a listener for the event.

Parameters

- **cls** (*class*) – The class of events for which the listener is registered.
- **func** (*callable*) – A callable to be called when the event is emitted. The function should expect one argument which is the event instance which was emitted.
- **filter** (*dict*) – Filters to match for before calling the listener. The listener is called only when the event matches all of the filter .

  Filter specification:

  - key: string which is name of an attribute of the event instance.
  - value: the value of the specified attribute.
- **priority** (*int*) – The priority of the listener. Higher priority listeners are called before lower priority listeners.

Note

Reconnecting an already connected listener will disconnect the old listener. This may have rammifications in changing the filters and the priority.

The filtering is added so that future optimizations can be done on specific events with large number of handlers. For example there should be a fast way to filter key events to specific listeners rather than iterating through all listeners.

disable(*cls*)
Disable the event from generating notifications.

Parameters **cls** (*class*) – The class of events which we want to disable.

disconnect(*cls*, *func*)
Disconnects a listener from being notified about the event'

Parameters

- **cls** (*class*) – The class of events for which the listener is registered.
- **func** (*callable*) – The callable which was registered for that class.

Raises **KeyError** - if *func* is not already connected.

emit(*event*, *block=True*)
Notifies all listeners about the event with the specified arguments.

Parameters

- **event** (instance of `BaseEvent`) – The `BaseEvent` instance to emit.
- **block** (*bool*) – Whether to block the call until the event handling is finished. If block is False, the event will be emitted in a separate thread and the thread will be returned, so you can later query its status or do `wait()` on the thread.

Note

Listeners of superclasses of the event are also called. Eg. a `BaseEvent` listener will also be notified about any derived class events.

enable(*cls*)
Enable the event again to generate notifications.

Parameters **cls** (*class*) – The class of events which we want to enable.

is_enabled(*cls*)
Check if the event is enabled.

Parameters **cls** (*class*) – The class of events which we want check the status of.

Progress Events

Events and helpers for managing progress indicators

class encore.events.progress_events.**ProgressManager**(*event_manager=None*, *source=None*, *operation_id=None*, *message='Performing operation'*, *steps=-1*, ***kwargs*)
Utility class for managing progress events

This class provides a context manager that will probably be sufficient in most use cases. The standard method of invoking it will be something like:

```
with ProgressManager(event_manager, source, id, "Performing operation", steps) as progress:
    for step in range(steps):
        ... do work ...
        progress(step)
```

This pattern guarantees that the appropriate Start and Stop events are always emitted, even if there is an exception.

If finer-grained control is needed, the class also provides start(), step() and stop() methods that can be invoked in when required. In particular, this pattern may be useful for more fine-grained exception reporting:

```
progress = ProgressManager(event_manager, source, id, "Performing operation", steps)
progress.start()
try:
    for step in range(steps):
        ... do work ...
        progress(step)
except ... as exc:
    progress.end(message='Failure mode 1', end_state='warning')
except ... as exc:
    progress.end(message='Failure mode 2', end_state='error')
except Exception as exc:
    progress.end(message=str(exc), end_state='exception')
else:
    progress.end(message='Success', end_state='normal')
```

StartEventType
(ProgressStartEvent subclass) The actual event class to use when emitting a start event. The default is *ProgressStartEvent*, but subclasses may choose to override.

StepEventType
(ProgressStepEvent subclass) The actual event class to use when emitting a step event. The default is *ProgressStepEvent*, but subclasses may choose to override.

EndEventType
(ProgressEndEvent subclass) The actual event class to use when emitting an end event. The default is *ProgressEndEvent*, but subclasses may choose to override.

__init__(*event_manager=None*, *source=None*, *operation_id=None*, *message='Performing operation'*, *steps=-1*, ***kwargs*)
Create a progress manager instance

Parameters

- **event_manager** (*EventManager instance*) – The event manager to use when emitting events.
- **source** (*any*) – The object that is the source of the events.
- **operation_id** (*any*) – The unique identifier for the operation.
- **message** (*string*) – The default message to use for events which are emitted.
- **steps** (*int*) – The number of steps. If this is not known, use -1.

end(*message=None*, *exit_state='normal'*, ***extra_kwargs*)
Emit a step event

By default creates an instance of *StepEventType* with the appropriate attributes.

Parameters

- **message** (*str*) – The message to be passed to the event's constructor. By default will use `self.message`.
- **exit_state** (one of `normal`, `warning`, `error` or `exception`) – The exit_state of the event.
- **extra_kwargs** (*dict*) – Additional arguments to be passed through to the event's constructor.

start (***extra_kwargs*)
Emit a start event

By default creates an instance of *StartEventType* with the appropriate attributes.

Parameters **extra_kwargs** (*dict*) – Additional arguments to be passed through to the event's constructor.

step (*message=None*, *step=None*, ***extra_kwargs*)
Emit a step event

By default creates an instance of *StepEventType* with the appropriate attributes.

Parameters

- **message** (*str*) – The message to be passed to the event's constructor. By default will use `self.message`.
- **step** (*int*) – The step number. By default keeps an internal step count, incremented each time this method is called.
- **extra_kwargs** (*dict*) – Additional arguments to be passed through to the event's constructor.

class `encore.events.progress_events.`**ProgressEvent** (*source=None*, ***kwargs*)
Abstract base class for all progress events

This class is provided so that listeners can easily listen for any type ProgressEvent.

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the operation being performed.

class `encore.events.progress_events.`**ProgressStartEvent** (*source=None*, ***kwargs*)
Event emitted at the start of an operation

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the operation being performed.

steps
(int) The number of steps in the operation. If unknown or variable, use -1.

class `encore.events.progress_events.`**ProgressStepEvent** (*source=None*, ***kwargs*)
Event emitted periodically during an operation

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the state of the operation being performed.

step
(int) The count of the step. If unknown, use -1.

class encore.events.progress_events.**ProgressEndEvent**(*source=None*, ***kwargs*)
Event emitted at the end of an operation

operation_id
A unique identifier for the operation that is finished.

message
(string) A human-readable describing the state of the operation that ended.

exit_state
(string) A constant describing the end state of the operation. One of `normal`, `warning`, `error` or `exception`.

3.1.2 Indices and tables

- genindex
- modindex
- search

3.1.3 License

3.1.4 util.human_date module

3.1.5 concurrent.futures.enhanced_thread_pool_executor

3.2 Storage

The `encore.storage` package provides an abstract API for key-value stores, as well as some reference implementations, utilities, and building-blocks for creating more complex stores from simple ones.

The API is both agnostic to the type of data being stored and the underlying data storage medium. Being agnostic to the type of data permits the API to be used in appropriate situations other than the ones that we currently envision for current Enthought projects, while being agnostic to the underlying storage mechanism permits the same code to be used no matter how the data is stored - whether in memory, a filesystem, a web service, or a SQL or NoSQL database - permitting greater flexibility in deployment depending on user needs.

All abstractions are leaky, so we don't anticipate that this API will cover all possible functionality that a data store could provide, but the hope is that the API provides a common language for the most fundamental operations, and a baseline which can be extended as we find more commonalities in the data stores that we develop.

3.2.1 Contents

Key-Value Store Concepts

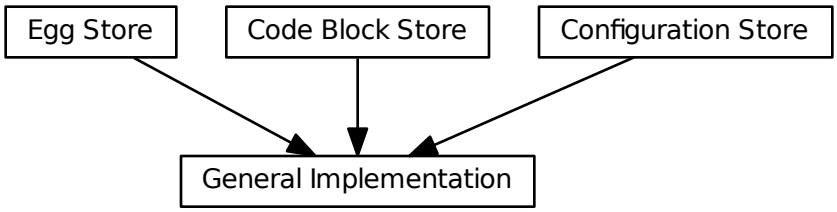
The key-value store API exposes an interface on top of whatever backend implementation is used by subclasses. This permits code which requires access to a key-value store to use it in a uniform way without having to care about where and how the data is stored. The API is also agnostic about what is being stored, and so while the key use case is for egg repositories, potentially any data values can be stored in the key-value store.

Discussion

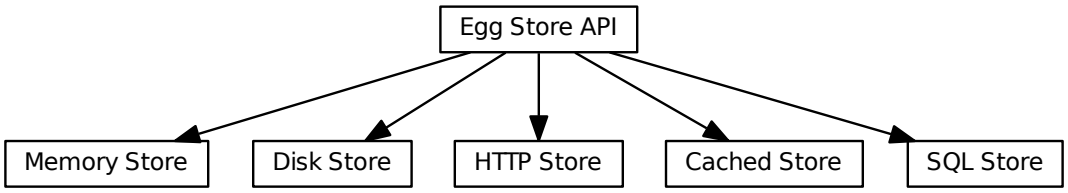
We have seen a common need throughout client and internal development efforts at Enthought for various ways of persistently storing data and associated metadata and making it available within the applications we write. Over the years, Enthought has implemented a number of different storage systems with similar general functionality; sometimes even with multiple ones in the same project. The particular motivation which has prompted the creation of the storage API, and the initial use-case is a refactor of the Enstaller project to provide a cleaner set of internal APIs.

When faced with a problem like this, it is tempting to start from the implementation level and build a solution to the problem at hand (eg. "I need to store eggs, so I build an egg store"). In generalizing, it is then even more tempting to try to replace existing implementations with a more generic implementation (eg. "I need to store eggs and apps and code blocks both locally and remotely, so I build a NoSQL-backed data store server which I can run locally if I need to").

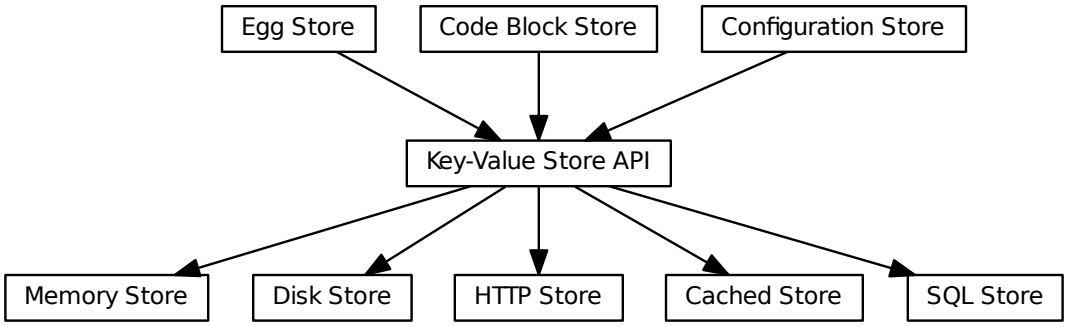
However any generic implementation must make trade-offs, and the trade-offs may end up being inappropriate for particular use cases (eg. "Now I need to have an in-memory code-block store for efficiency, and my store's optimized for remote access...").



Alternatively, it is tempting to start from the task that is being attempted (eg. "I need to store eggs, so I build an egg store") and then generalize the implementation (eg. "Now I need to store eggs remotely, so I'll build a remote egg store, but at least I'll use the same API"). However, again the specifics of the implementation may make it inappropriate for particular use cases (eg. "Now I need a code-block store, so I'll wrap up my code-blocks as eggs and use an egg store... but now they're 100 times bigger than they need to be...")



This storage API instead to simply provides an API through which data providers and data consumers can talk. The API deliberately makes no assumptions about what is being stored (eg. eggs vs. code blocks) but also makes no assumptions about how the data is stored (eg. in memory, on disk, in a database, through a remote server). This allows developers to re-use code more efficiently by permitting them to choose the pieces that make sense for their particular use case (eg. "I need an in-memory code-block store, so I'll take my general code-block store logic which uses the API, and my in-memory store logic which implements the API, and join them together"). Not every combination may make sense (an in-memory egg store is probably a bad idea, for example), but the ability to pick and choose allows a great deal of flexibility.



This also reflects the reality that developers frequently do not have complete freedom to choose the best possible data store solution due to external constraints. By writing to the storage API, you have the opportunity to more easily re-use components, as well as the possibility of later replacing the sub-optimal solution with a better one.

This approach also allows developers to write general connectors, adapters and other building blocks for repositories which only use the API and don't care about what is being stored. This should permit fast prototyping of functionality, if nothing else, but in many cases this approach may be good enough for production code. For example, a generic joined store could be written which takes a list of other stores which implement the API and when asked for data from the store asks each store in sequence for the data until it finds what is requested. To code using the joined store, it appears just like any other store, and the joined store doesn't care how the stores it joins are implemented.

Keys

The keys of the key-value store are strings, and the key-value store API makes no assumptions about what the strings represent or any structure they might have. In particular keys are assumed to be case sensitive and may include arbitrary characters, so key-value store implementations should be careful to handle any issues which may arise if the underlying data store is case insensitive and has special characters which need to be escaped.

Each key has associated with it a collection of metadata and some binary data. The key-value store API makes no assumptions about how the metadata and data is serialized.

Values

The values stored in the key-value store consist of two parts, a binary data stream and a metadata dictionary. These are encapsulated into a light-weight data-structure which can hold additional implementation-specific information.

In particular, implementations should expose attributes or properties 'size', 'created' and 'modified' which proved the number of bytes in the data stream, the creation time of the key, and the most recent modification time of the key. These additional attributes are primarily provided for internal use and to assist composition and replication of key-value stores.

The Value should contain enough information to extract the data and metadata, but does not have to actually open those resources until they are requested.

For writable repositories, data should be supplied to keys via a Value subclass, if possible. This allows copying between repositories using code like:

```
repo1.set(key, repo1.get(key))
```

or copying between keys with code like:

```
repo.set(key1, repo.get(key2))
```

Since files are likely to be common targets for extracting data from values, or sources for data being stored, the key-value store API provides utility methods `to_file()` and `from_file()`. Simple default implementations of these methods are provided, but implementations of the key-value store API may be able to override these to be more efficient, depending on the nature of the back-end data store.

For backwards compatibility, value objects express an API that makes them appear as a 2-tuple of (data, metadata).

Data

The binary data stored in the values is presented through the key-value store API as file-like objects which implement at least read(), close(), __enter__() and __exit__() methods as well as having attributes which provide some amount of information about the stream, such as length, last modification time, creation time, and so forth. Particular backends may choose to provide additional attributes or implement additional methods as needed.

Frequently this will be a wrapper around a standard file, StringIO object, a urllib file-like object or other wrapper about a socket. The read() method should accept an optional number of bytes to read, so that buffered reads can be performed.

The key-value store API gives no special meaning to the bytes stored in the value. However care should be taken that it is in fact bytes being stored, and not a (possibly unicode) string; in particular, if an actual file is being used it should be opened in binary mode.

Metadata

Metadata should be representable as a dictionary whose keys are valid Python identifiers, and whose values can be serialized into reasonable human-readable form (basically, you should be able to represent the dictionary as JSON, XML, YAML, or similar in a clear and sane way, because some underlying datastore *will*).

Metadata can be retrieved via the `get_metadata()` method or as the second element of the tuple returned by `get()`. Metadata can be set using `set()` or `set_metadata()` and existing metadata can be modified using `update_metadata()` (similarly to the way that the `update()` method works for dictionaries).

There is nothing that ensures that metadata and the corresponding data are synchronised for a particular object. It is up to the user of the API to ensure that the metadata for stored data is correct.

We currently make no assumptions about the metadata keys, but we expect conventions to evolve for the meanings and format of particular keys. Given that this is generally thought of as a repository for storing eggs, the following metadata keys are likely to be available:

type The type of object being stored (package, app, patch, video, etc.).

name The name of the object being stored.

version The version of the object being stored.

arch The architecture that the object being stored is for.

python The version of Python that the object being stored is for.

ctime The creation time of the object in the repository in seconds since the Unix Epoch.

mtime The last modification time of the object in the repository in seconds since the Unix Epoch.

size The size of the binary data in bytes.

Note that there is a difference in intent between the information stored in the metadata and the attributes on the value object: value object attributes are controlled by the key-value store implementation, whereas metadata are completely arbitrary from the point of view of the key-value store and are completely up to the user code as to what information is stored.

Connecting and Disconnecting

Before a store can be used, its `connect()` method must be called to allow any long-lived resources to be allocated and prepared for use, and to optionally handle any authentication that might be required.

Conversely, the store's `disconnect()` method should be called when code is done with the store, allowing it to release any long-lived resources.

Querying

A very simple querying API is provided by default. The `query()` method simply takes a collection of keyword arguments and interprets them as metadata keys and values. It returns all the keys and corresponding metadata that match all of the supplied arguments. `query_keys()` does the same, but only returns the matching keys.

Subclasses may choose to provide more sophisticated querying mechanisms.

Transactions

The base abstract key-value store has no notion of transactions, since we want to handle the read-only and simple writer cases efficiently. However, if the underlying storage mechanism has the notion of a transaction, this can be encapsulated by writing a context manager for transactions. The `transaction()` method returns an instance of the appropriate context manager.

Events

All implementations should have an event manager attribute, and may choose to emit appropriate events. This is of particular importance during long-running interactions so that progress can be displayed. This also provides a mechanism that an implementation can use to inform listeners that new objects have been added, or the store has been otherwise modified.

Notes For Writing An Implementation

Metadata is really an index In terms of traditional database design, things that you are exposing in metadata are really indexed columns. If you are implementing a store which needs fast querying, you may want to look at how traditional databases do indexing to guide your data structure choices.

Determine the Single Points of Truth Every piece of data should have a single point of truth - a canonical place which holds the correct value. This is particularly true for metadata.

Testing There are standard test suites that can validate that a store is working as expected by the API. When writing an implementation of the API, you can subclass the tests and write appropriate setUp and tearDown methods that will put the store into the correct state.

Usage

The key-value store API gives a common API that can be used with a variety of different backends to provide a consistent interface for storage. If used correctly you can swap out the backend used with little or no modification of the user code.

Creating and Connecting

Before you use a store, you need to create an instance of the appropriate type, and then connect to it, possibly authenticating if that is required. For example, the following connects to a read-only remote store via HTTP, using HTTP Authentication:

```
from encore.events.api import EventManager
from encore.storage.static_url_store import StaticURLStore

event_manager = EventManager()
store = StaticURLStore(event_manager, 'http://localhost:8080/', 'data', 'index.json')
store.connect(credentials={'username': 'alibaba', password: 'Open Sesame'})
```

At this point the store is ready to use. You can check to see whether the store has connected using the *is_connected()* method. When you are finished with a store, you should call its *disconnect()* method to allow it to cleanly release any resources it may be using, such as database connections.

Reading

To read from a store, you use one of the *get()* methods:

```
value = store.get('my_document')
datastream = value.data
metadata = value.metadata
```

In this case datastream is a file-like object that streams bytes:

```
data = datastream.read()
print data
```

More likely you will have used some sort of serialization format like XML, JSON or YAML to store your data in the document, so instead you can do:

```
import json
data = json.load(datastream)
```

If the data is raw bytes to store into a numpy array, you can do something like this:

```
import numpy
data = datastream.read()
dtype = numpy.int32
size = len(data)/dtype().nbytes
arr = numpy.empty(shape=size, dtype=dtype)
arr.data[:] = data
```

The `read()` method supports buffered reads if your data is larger than would comfortably fit into memory.

If you need to support random-access streaming, the value API also supports a `range(start, end)()` method that return the requested bytes as a readable stream.

The metadata stores auxilliary information about the data that is stored in the key. It is a dictionary of reasonably serializable values (frequently it will serialize to JSON or similar format):

```
print 'Document title:', metadata['title']
print 'Document author:', metadata['author']
print 'Document encoding:', metadata['encoding']

# checksum
import hashlib
assert hashlib.sha1(document.read()).digest() == metadata['sha1']
```

What metadata is stored is completely dependent on the use-case for the key-value store: the key-value store makes no assumptions.

If you try to read a key which doe not exist, then the store will raise a KeyError. If you want to see whether or not a particular key is populated, you can use the *exists()* method.

Frequently you will only be interested in the data or the metadata, not both. For these cases there are methods *get_data()* and *get_metadata()* which return the appropriate entities. For metadata, if you are only interested in the values of some of the dictionary keys, you can supply an additional argument `select` which will restrict the returned keys to this subset of all the keys:

author_info = store.get_metadata('document', select=['author', 'organization'])

It is very common that you either want to extract the stream of bytes from a value into a Python bytes object (ie. a string in Python 2, as opposed to unicode) or into a file on the local filesystem. Two utility methods *to_file()* and *to_bytes()* are provided which perform these operations. If the data source is larger than will comfortably fit into memory (particularly for *to_file()*) you can supply an optional buffer size:

```
store.to_file('document', 'local_document.txt', buffer=8096)
```

Querying

Frequently you want to find keys whose metadata match certain criteria. The key-value store API gives a simple query mechanism that permits this sort of matching:

```
for key, metadata in store.query(author='alibaba', organization='40 Thieves'):
    print key, ':', metadata['title']
```

This will print the key and title of all documents which have an `author` key with value `'alibaba'` and an `organization` key with value `'40 Thieves'`. The current API only permits querying for exact matches and matching all of the query terms. More complex queries would need to be performed on an ad-hoc basis on top of this API.

If all the user is concerned with is which keys match, there is an alternative method *query_keys()*:

```
for key in store.query_keys(author='alibaba', organization='40 Thieves'):
    print key
```

To iterate over all the keys in a store, you can simply call *query_keys()* with no arguments:

```
for key in store.query_keys():
    print key
```

Finally, as a useful utility, you can use glob-style matching on the keys using the *glob()* method:

```
for key in store.glob('*.jpg'):
    print key
```

Writing

Most, but not all, stores also allow you to write data to keys. The basic method is *set()* which is the inverse of *get()*. It expects a file-like object with a `read()` method that can do buffering, and a dictionary of metadata as arguments:

```
from cStringIO import StringIO

data = StringIO("Hello World")
metadata = {'title': "Greeting", 'author': 'alibaba'}
store.set('hello', (data, metadata))
```

As with reading, there are methods *set_data()* and *set_metadata()* that permit you to set just one of the two parts of the value, and there are utility methods *from_bytes()* and *from_file()* that populate the data of a key from either a byte string or a binary file. The latter two methods do not set any metadata: that must be done manually if needed.

If you want to add to the metadata without overwriting it, there is a convenience method *update_metadata()* method that will update the metadata dictionary in mych the same way that the standard Python dictionary's `update` method works.

You can delete a key with the *delete()* method:

```
store.delete('hello')
```

Transactions

The key-value store API does not assume that the underlying storage mechanism has a notion of transactions, but if it does then it can be supported by the key-value store. Transactions are handled by context managers and the with statement:

```
with store.transaction('Setting some values'):
    store.set('key1', (data1, metadata1))
    store.set('key2', (data2, metadata2))
```

If any exception were to occur in the with statement, the context manager will ensure that the transaction gets rolled back. Otherwise the transaction will be committed when the with statement finishes.

Transactions are re-entrant, so it is safe to do the following:

```
def add_keypair(keypair):
    with store.transaction('Adding keypair'):
        store.set(keypair.key1, (keypair.data1, keypair.metadata1))
        store.set(keypair.key2, (keypair.data2, keypair.metadata2))
```

```
def add_many_keypairs(keypairs):
    with store.transaction('Adding many keypairs'):
        for keypair in keypairs:
            add_keypair(keypair)
```

The transaction in the function is effectively ignored, with only the outermost transaction applying.

The "Multi" Methods

For convenience there are a collection of methods prefixed by "multi", such as *multiget()* and *multiset_data()*, which perform the specified operations on a collection of keys at once. If transactions are available, then these will be done as a single transaction.

Events

The various stores use the Encore event system, which is why the stores must be supplied with a reference to an EventManager instance. The events which are emitted are referenced in the documentation for each method.

Key-Value Store API

This module specifies the key-value store API for the various package management and installation systems that are in use at Enthought and our clients.

The key-value store API exposes an interface on top of whatever backend implementation is used by subclasses. This permits code which requires access to a key-value store to use it in a uniform way without having to care about where and how the data is stored. The API is also agnostic about what is being stored, and so while the key use case is for egg repositories, potentially any data values can be stored in the key-value store.

class encore.storage.abstract_store.**Value**

Abstract base class for file-like objects used by Key-Value stores

size [int] The size of the data in bytes, or None if a continuous stream or unknown.

created [timestamp] The creation time of the key as a floating point UTC timestamp in seconds after the Unix Epoch.

modified [timestamp] The modification time of the key as a floating point UTC timestamp in seconds after the Unix Epoch.

data

The byte stream of data contained in the value

iterdata(*buffer_size=1048576*, *progress=None*)

Return an iterator over the data stream

metadata

The metadata dictionary of the value

permissions

The permissions dictionary of the value

This is only available if the user has ownership privileges for the key. Because different stores have different permission conventions, this will not be used when setting a value.

range(*start=None*, *end=None*)

Return a stream with a range of bytes from the data

class encore.storage.abstract_store.**AbstractReadOnlyStore**

Abstract base class for read-only Key-Value Store API

This class implements some of the API so that it can be used with super() where appropriate.

event_manager
Every store is assumed to have an event_manager attribute which implements the *BaseEventManager* API.

connect(*credentials=None*)
Connect to the key-value store, optionally with authentication

This method creates or connects to any long-lived resources that the store requires.

Parameters **credentials** – An object that can supply appropriate credentials to to authenticate the use of any required resources. The exact form of the credentials is implementation-specific, but may be as simple as a `(username, password)` tuple.

Raises **AuthorizationError** - If the credentials are not valid, this error will be raised.

disconnect()
Disconnect from the key-value store

This method disposes or disconnects to any long-lived resources that the store requires.

exists(*key*)
Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

get(*key*)
Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **value** (instance of Value) - An instance of a Value subclass which holds references to the data, metadata and other information about the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)
Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_data_range(*key*, *start=None*, *end=None*)
Retrieve a partial stream from a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **start** (*int or None*) – The first byte to return
- **end** (*int or None*) – The last byte of to return

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)
Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

glob(*pattern*)
Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()
Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)
Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

query (*select=None*, ***kwargs*)

Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys (***kwargs*)

Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

to_bytes (*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a bytes object. The default implementation uses the get() method together with chunked reads from the returned data stream and join.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Returns **bytes** - The contents of the file-like object as bytes.

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to extracting the data.

- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is extracted.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after extracting the data.

to_file(*key*, *path*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a file.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a path in the filesystem. The default implementation uses the get() method together with chunked reads from the returned data stream to the disk.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to disk.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to disk.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to disk.

class encore.storage.abstract_store.**AbstractStore**

Abstract base class for Key-Value Store API

This class implements some of the API so that it can be used with super() where appropriate.

event_manager

Every store is assumed to have an event_manager attribute which implements the *BaseEventManager* API.

delete(*key*)

Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

from_bytes(*key*, *data*, *buffer_size=1048576*)

Efficiently store a bytes object as the data associated with a key.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a bytes object to the underlying data store. The default implementation uses the set() method together with a cStringIO.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

from_file(*key*, *path*, *buffer_size=1048576*)

Efficiently read data from a file into a key in the key-value store.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a path in the filesystem to the underlying data store. The default implementation uses the set() method together with chunked reads from the disk which are fed into the data stream.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

multiset(*keys*, *values*, *buffer_size=1048576*)

Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)
Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)
Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)
Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

set (*key*, *value*, *buffer_size=1048576*)

Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*instance of Value*) – An instance of a Value subclass.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data (*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.

- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

transaction(*notes*)

Provide a transaction context manager

Implementations which have no native notion of transactions may choose not to implement this.

This method provides a context manager which creates a data store transaction in its __enter__() method, and commits it in its __exit__() method if no errors occur. Intended usage is:

```
with repo.transaction("Writing data..."):
    # everything written in this block is part of the transaction
    ...
```

If the block exits without error, the transaction commits, otherwise the transaction should roll back the state of the underlying data store to the start of the transaction.

Parameters **notes** (*string*) – Some information about the transaction, which may or may not be used by the implementation.

Returns **transaction** (context manager) - A context manager for the transaction.

Events

- **StoreTransactionStartEvent** - This event should be emitted on entry into the transaction.
- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreTransactionEndEvent** - This event should be emitted on successful conclusion of the transaction, before any Set or Delete events are emitted.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set during the transaction.
- **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key for all deleted keys.

update_metadata(*key*, *metadata*)

Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

class encore.storage.abstract_store.**AbstractAuthorizingStore**

Abstract base class for Key-Value Store API with permissioning

This class implements some of the API so that it can be used with super() where appropriate.

Permission information is available only to authenticated users who are designated as owners of a particular key. Permissions are simply strings representing some right that the store allows, the only required permission being 'owned'.

Each permission has a set of tags which are granted that permission. A tag represents a user, group or role that will be granted that permission. The meaning of tags is also store dependent: a filesystem-based store may have tags for 'user', 'group' and 'other'; while a web-based store may derive its tags from a role-based authentication system.

event_manager

Every store is assumed to have an event_manager attribute which implements the *BaseEventManager* API.

get_permissions(*key*)

Return the set of permissions the user has

Parameters **key** (*str*) – The key for the resource which you want to know the permissions.

Returns **permissions** (dict of str: set of str) - A dictionary whose keys are the permissions and values are sets of tags which have that permission.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

set_permissions(*key*, *permissions*)

Set the permissions on a key the user owns

Parameters

- **key** (*str*) – The key for the resource which you want to know the permissions.
- **permissions** (*dict of str: set of str*) – A dictionary whose keys are the permissions and values are sets of tags which have that permission. There must be an 'owned' permission with at least one tag.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

update_permissions(*key*, *permissions*)
Add permissions on a key the user owns

The tags provided in the permissions dictionary will be added to the existing set of tags for each permission.

Parameters

- **key** (*str*) – The key for the resource which you want to know the permissions.
- **permissions** (*dict of str: set of str*) – A dictionary whose keys are the permissions and values are sets of tags which have that permission.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

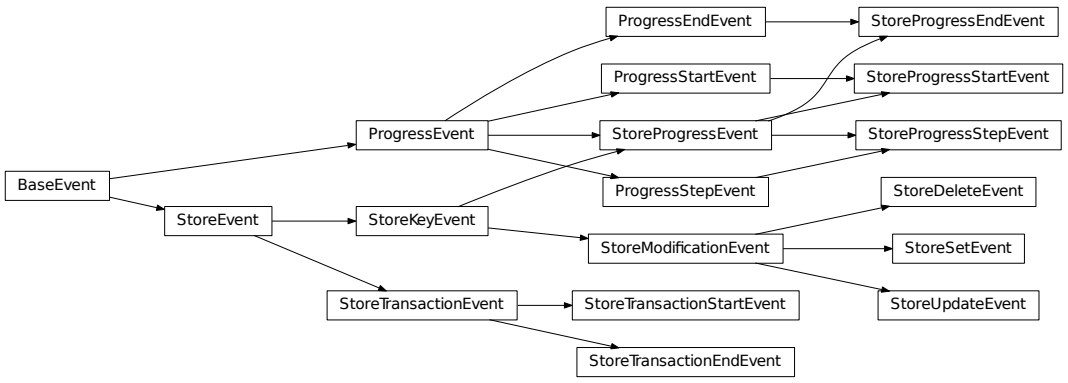
user_tag
A tag that represents the user

Events

The Storage API generates events using the Encore Event API. This module defines the event classes that are required aby the abstract API.

Event Inheritance Diagram

The following diagram shows the inheritance heirarchy of the various Event subclasses defined in this module. When listening for events, you may want to listen on appropriate superclasses.



Storage Events This module contains asbtract and concrete Event subclasses that support the Storage API.

class encore.storage.events.**StoreEvent**(*source=None*, ***kwargs*)
An abstract base class for events generated by a Key-Value Store

source
(Store instance) The key-value store which generated the event.

class encore.storage.events.**StoreKeyEvent**(*source=None*, ***kwargs*)
An abstract base class for events related to a particular key in the store. This should provide the key and metadata (if available) of the modified key.

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

class encore.storage.events.**StoreModificationEvent**(*source=None*, ***kwargs*)
An abstract base class for modification events generated by a Key-Value Store

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

action
(string) The modification action that was performed. One of 'set', 'update' or 'delete'.

class encore.storage.events.**StoreSetEvent**(*source=None*, ***kwargs*)
An event generated when a value is set into a Key-Value Store

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

action
('set') The modification action that was performed.

class encore.storage.events.**StoreUpdateEvent**(*source=None*, ***kwargs*)
An event generated when a value is updated into a Key-Value Store

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

action
('update') The modification action that was performed.

class encore.storage.events.**StoreDeleteEvent**(*source=None*, ***kwargs*)
An event generated when a value is deleted into a Key-Value Store

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

action
('delete') The modification action that was performed.

class encore.storage.events.**StoreProgressEvent**(*source=None*, ***kwargs*)
Abstract base class for ProgressEvents generated by a Key-Value Store

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the operation being performed.

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

class encore.storage.events.**StoreProgressStartEvent**(*source=None*, ***kwargs*)

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the operation being performed.

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

steps
(int) The number of steps in the operation. If unknown or variable, use -1.

class encore.storage.events.**StoreProgressStepEvent**(*source=None*, ***kwargs*)

operation_id
A unique identifier for the operation being performed.

message
(string) A human-readable describing the state of the operation being performed.

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

step
(int) The count of the step. If unknown, use -1.

class encore.storage.events.**StoreProgressEndEvent**(*source=None*, ***kwargs*)

operation_id
A unique identifier for the operation that is finished.

message
(string) A human-readable describing the state of the operation that ended.

key
(string) The key which is involved in the event.

metadata
(dict) The metadata of the key which is involved in the event.

exit_state
(string) A constant describing the end state of the operation. One of `normal`, `warning`, `error` or `exception`.

Utils

Utilities for key-value stores.

File-like Interface Utilities

These utilities help with the management of file-like objects. In particular *buffer_iterator()* is of particular use, as it produces an iterator which generates chunks of bytes in the file-like object which permits memory-efficient streaming of the data. This is preferred over reading in all the data and then processing it if the data is even moderately big.

The *BufferIteratorIO* class is a class whick provides a file-like API around a buffer iterator. This is particularly useful for Stores which wrap another store and implementing streaming filters on the data.

class encore.storage.utils.**BufferIteratorIO**(*iterator*)
A file-like object based on an iterable of buffers

This takes an iterator of bytes objects, such as produced by the buffer_iterator function, and wraps it in a file-like interface which is usable with the store API.

This uses less memory than a StringIO, at the cost of some flexibility.

Parameters **iterator** (*iterator of bytes objects*) – An iterator that produces a bytes object on each iteration.

read(*buffer_size=1048576*)
Read at most buffer_size bytes, returned as a string.

encore.storage.utils.**buffer_iterator**(*filelike*, *buffer_size=1048576*, *progress=None*, *max_bytes=None*)
Return an iterator of byte buffers

The buffers of bytes default to the provided buffer_size. This is a useful method when copying one data stream to another.

Parameters

- **filelike** (*a file-like object*) – An object which implements the `read(buffer_size)()` method.
- **buffer_size** (*int*) – The number of bytes to read at a time.
- **progress** (*callable*) – A callback for progress indication. A StoreProgressManager instance inside a `with` block would be appropriate, but anthing that takes a *step* parameter which is the total number of bytes read so far will work.
- **max_bytes** (*int*) – The maximum number of bytes to return.

encore.storage.utils.**tee**(*filelike*, *n=2*, *buffer_size=1048576*)
Clone a filelike stream into n parallel streams

This uses itertools.tee and buffer iterators, with the corresponding cautions about memory usage. In general it should be more memory efficient than pulling everything into memory.

Parameters

- **filelike** (*a file-like object*) – An object which implements the `read(buffer_size)()` method.

- **n** (*int*) – The number of filelike streams to produce.
- **buffer_size** (*int*) – The number of bytes to read at a time.

Transaction Support

These are two simple context managers for transactions. The *DummyTransactionContext* should be used by Store implementations which have no notion of a transaction. The *SimpleTransactionContext* is a complete transaction manager for implementations with begin/commit/rollback semantics.

class encore.storage.utils.**DummyTransactionContext**
A dummy class that can be returned by stores which don't support transactions

This class guarantees that there is only one transaction object for each store instance.

Parameters **store** (*key-value store instance*) – The store that this transaction context is associated with.

class encore.storage.utils.**SimpleTransactionContext**
A simple class that adds support for simple transactions

This is a base class that ensures transactions are appropriately handled in terms of nesting and event generation. Subclasses should override the start, commit and rollback methods to perform appropriate implementation-specific actions.

This class correctly handles nested transactions by ensuring that each store has precisely one active transaction context and by tracking the number of times the context has been entered and exited. The transaction is only committed once the top-level context has exited.

Parameters **store** (*key-value store instance*) – The store that this transaction context is associated with.

begin()
Begin a transaction

By default, this calls the store's _begin_transaction method. Override in subclasses if you need different behaviour.

commit()
Commit a transaction

By default, this calls the store's _commit_transaction method. Override in subclasses if you need different behaviour.

rollback()
Roll back a transaction

By default, this calls the store's _rollback_transaction method. Override in subclasses if you need different behaviour.

Event Support

class encore.storage.utils.**StoreProgressManager**(*event_manager=None*, *source=None*, *operation_id=None*, *message='Performing operation'*, *steps=-1*, ***kwargs*)
encore.events.progress_events.ProgressManager subclass that generates *encore.storage.events.StoreProgressEvent* instances

EndEventType
alias of StoreProgressEndEvent

StartEventType
alias of StoreProgressStartEvent

StepEventType
alias of StoreProgressStepEvent

Implementations

Memory Store

This is a simple implementation of the key-value store API that lives entirely in memory. Data and metadata are stored in dictionaries. This is not optimized in any way to reduce memory usage.

This class is provided in part as a sample implementation of the API.

class encore.storage.dict_memory_store.**DictMemoryStore**
Dictionary-based in-memory Store

This is a simple implementation of the key-value store API that lives entirely in memory. This uses a dictionary of StringValue objects to store all relevant information about an object - data and metadata are stored in private attributes.

The streams returned by data methods are cStringIO.StringIO objects.

Parameters **event_manager** – An object which implements the *BaseEventManager* API.

connect(*credentials=None*)
Connect to the key-value store

Parameters **credentials** (*None*) – This store does not authenticate, and has no external resources, so credentials are ignored.

delete(*key*)
Delete a key from the repsository.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

disconnect()
Disconnect from the key-value store

This store does not authenticate, and has no external resources, so this does nothing

exists(*key*)
Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

from_bytes(*key*, *data*, *buffer_size=1048576*)
Efficiently read data from a bytes object into a key in the key-value store.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – This is ignored.

from_file(*key*, *path*, *buffer_size=1048576*)

Efficiently read data from a file into a key in the key-value store.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – This is ignored.

get(*key*)

Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)

Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

get_metadata(*key*, *select=None*)

Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the metadata_keys argument.

Raises **KeyError** - This will raise a key error if the key is not present in the store, and if any metadata key is requested which is not present in the metadata.

glob(*pattern*)

Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()
Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)
Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiset(*keys*, *values*, *buffer_size=1048576*)
Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)

Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)

Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)

Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

query(*select=None*, ***kwargs*)

Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of keys, metadata tuples where metadata matches all the specified values for the specified metadata keywords.

query_keys(***kwargs*)

Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)

Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data(*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

If the key does not already exist, it tacitly creates an empty metadata object.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata. If the key does not already exist, it tacitly creates an empty data object.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

to_bytes(*key*, *buffer_size=1048576*)
Efficiently store the data associated with a key into a bytes object.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – This is ignored.

to_file(*key*, *path*, *buffer_size=1048576*)
Efficiently store the data associated with a key into a file.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – This is ignored.

transaction(*notes*)
Provide a transaction context manager

This class does not support transactions, so it returns a dummy object.

Parameters **notes** (*string*) – Some information about the transaction, which is ignored by this implementation.

update_metadata(*key*, *metadata*)
Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Sqlite Store

This is a simple implementation of the key-value store API that lives in a sqlite database. Each key is stored in a row which consists of the key, index columns, metadata and data. The index columns are a specified subset of the metadata that can be queried more quickly.

This class is provided in part as a sample implementation of the API.

class encore.storage.sqlite_store.**SqliteStore**(*location=':memory:'*, *table='store'*, *index='dynamic'*, *index_columns=None*)
Sqlite-based Store

The file-like objects returned by data methods are cStringIO objects.

Warning: The table name and metadata names used as index columns are not sanitized. To prevent SQL injection these should never be directly derived from user-supplied values. This is particularly important for indexed queries.

connect(*credentials=None*)
Connect to the key-value store

This connects to the specified location and creates the table, if needed. Sqlite has no notion of authentication, so credentials are ignored.

delete(*key*)
Delete a key from the repsository.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

disconnect()
Disconnect from the key-value store

This clears the reference to the sqlite connection object, allowing it to be garbage-collected.

exists(*key*)
Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

from_bytes(*key*, *data*, *buffer_size=1048576*)
Efficiently read data from a bytes object into a key in the key-value store.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – This is ignored.

from_file(*key*, *path*, *buffer_size=1048576*)
Efficiently read data from a file into a key in the key-value store.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – This is ignored.

get(*key*)
Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)
Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)
Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the metadata_keys argument.

Raises **KeyError** - This will raise a key error if the key is not present in the store, and if any metadata key is requested which is not present in the metadata.

glob(*pattern*)
Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()
Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)

Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiset(*keys*, *values*, *buffer_size=1048576*)

Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)

Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)

Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)

Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

query(*select=None*, ***kwargs*)

Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of keys, metadata tuples where metadata matches all the specified values for the specified metadata keywords.

query_keys(***kwargs*)

Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)

Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

set_data(*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

to_bytes(*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – This is ignored.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

to_file(*key*, *path*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a file.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – This is ignored.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

transaction(*notes*)

Provide a transaction context manager

update_metadata(*key*, *metadata*)

Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

File System Store

This file defines a filesystem store. This stores data in a specified directory in a filesystem. Data files are stored in files with name key+'.data' and metadata files with name key+'.metadata'.

encore.storage.filesystem_store.**init_shared_store**(*path*, *magic_fname='.FSStore'*)

Create the magic file for the shared store. Useful to initialize the store for the first time.

Parameters

- **path** – The directory that will be used for the file store.
- **magic_fname** – The name of the magic file in that directory,

class encore.storage.filesystem_store.**FileSystemStore**(*path*, *magic_fname='.FSStore'*)

A store that uses a Shared file system to store the data/metadata.

__init__(*path*, *magic_fname='.FSStore'*)

Initializes the store given a path to a store.

Parameters

- **path** (*str:*) – A path to the root of the file system store.
- **magic_fname** – The name of the magic file in that directory,

connect(*credentials=None*)

Connect to the key-value store.

Parameters **credentials** – These are not used by default.

delete(*key*)

Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

disconnect()

Disconnect from the key-value store

This store does not authenticate, and has no external resources, so this does nothing

exists(*key*)

Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

from_bytes(*key*, *data*, *buffer_size=1048576*)

Efficiently store a bytes object as the data associated with a key.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a bytes object to the underlying data store. The default implementation uses the set() method together with a cStringIO.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

from_file(*key*, *path*, *buffer_size=1048576*)

Efficiently read data from a file into a key in the key-value store.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a path in the filesystem to the underlying data store. The default implementation uses the set() method together with chunked reads from the disk which are fed into the data stream.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

get(*key*)

Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)

Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)
Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

glob(*pattern*)
Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()
Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)
Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiset(*keys*, *values*, *buffer_size=1048576*)

Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)

Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.

- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)

Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)

Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

query(*select=None*, ***kwargs*)

Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is

not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys(***kwargs*)
Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)
Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data(*key*, *data*, *buffer_size=1048576*)
Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

to_bytes(*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a bytes object. The default implementation uses the get() method together with chunked reads from the returned data stream and join.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Returns **bytes** - The contents of the file-like object as bytes.

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to extracting the data.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is extracted.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after extracting the data.

to_file(*key*, *path*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a file.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a path in the filesystem. The default implementation uses the get() method together with chunked reads from the returned data stream to the disk.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to disk.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to disk.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to disk.

transaction(*notes*)

Provide a transaction context manager

This class does not support transactions, so it returns a dummy object.

update_metadata(*key*, *metadata*)

Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Static URL Store

This module contains the *StaticURLStore* store that communicates with a remote HTTP server which provides the actual data storage. This is a simple read-only store that can be run against a static HTTP server which provides a json file with all metadata and then serves data from URLs from another path. The metadata URL is polled periodically for updates.

A typical static server might be layed out as:

```
base_directory/
    index.json
    data/
        key1
        key2
        ...
```

class encore.storage.static_url_store.**StaticURLStore**(*root_url*, *data_path*, *query_path*, *poll=300*)

A read-only key-value store that is a front end for data served via URLs

All data is assumed to be served from some root url. In addition the store requires knowledge of two paths: a data prefix URL which is a partial URL to which the keys will be appended when requesting data, and a query URL which is a single URL which provides all metadata as a json encoded file.

For example, an HTTP server may store data at URLs of the form:

```
http://www.example.com/data/<key>
```

and may store the metadata at:

```
http://www.example.com/index.json
```

These would have a root url of "http://www.example.com/", a data path of "data/" and a query path of "index.json".

All queries are performed using urllib.urlopen, so this store can be implemented by an HTTP, FTP or file server which serves static files. When connecting, if appropriate credentials are supplied then HTTP authentication will be used when connecting the remote server

Warning: Since we use urllib without any further modifications, HTTPS requests do not validate the server's certificate.

Because of the limited nature of the interface, this store implementation is read only, and handles updates via periodic polling of the query prefix URL. This guarantees that the viewed data is always consistent, it just may not be current. Most of the work of querying is done on the client side using the cached metadata.

Parameters

- **event_manager** – An event_manager which implements the *BaseEventManager* API.
- **root_url** (*str*) – The base url that data is served from.
- **data_path** (*str*) – The URL prefix that the data is served from.
- **query_path** (*str*) – The URL that the metadata is served from.
- **poll** (*float*) – The polling frequency for the polling thread. Polls every 5 min by default.

connect(*credentials=None*, *proxy_handler=None*, *auth_handler_factory=None*)

Connect to the key-value store, optionally with authentication

This method creates appropriate urllib openers for the store.

Parameters

- **credentials** (*dict*) – A dictionary which has at least keys 'username' and 'password' and optional keys 'uri' and 'realm'. The 'uri' will default to the root url of the store, and 'realm' will default to 'encore.storage'.
- **proxy_handler** (*urllib.ProxyHandler*) – An optional urllib.ProxyHandler instance. If none is provided then urllib will create a proxy handler from the user's environment if needed.
- **auth_handler_factory** – An optional factory to build urllib authenticators. The credentials will be passed as keyword arguments to this handler's add_password method.

disconnect()

Disconnect from the key-value store

This method disposes or disconnects to any long-lived resources that the store requires.

exists(*key*)
Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

get(*key*)
Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store. This is the same type of filelike object returned by urllib's urlopen function.
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)
Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store. This is the same type of filelike object returned by urllib's urlopen function.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)
Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

glob(*pattern*)
Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()
Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)
Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

query(*select=None*, ***kwargs*)
Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys(***kwargs*)
Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

to_bytes(*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a bytes object. The default implementation uses the get() method together with chunked reads from the returned data stream and join.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Returns **bytes** - The contents of the file-like object as bytes.

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to extracting the data.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is extracted.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after extracting the data.

to_file(*key*, *path*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a file.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a path in the filesystem. The default implementation uses the get() method together with chunked reads from the returned data stream to the disk.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to disk.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to disk.

- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to disk.

update_index()
Request the most recent version of the metadata

This downloads the json file at the query_path location, and updates the local metadata cache with this information. It then emits events that represent the difference between the old metadata and the new metadata.

This method is normally called from the polling thread, but can be called by other code when needed. It locks the metadata index whilst performing the update.

Dynamic URL Store

This module contains the *DynamicURLStore* store that communicates with a remote HTTP server which provides the actual data storage. This is a store which implements the basic operations via HTTP GET, POST, PUT and DELETE commands as described in the class documentation.

The implementation relies on the third-party *requests* library to handle the HTTP operations.

class encore.storage.dynamic_url_store.**DynamicURLStore**(*base_url*, *query_url*, *url_format='{base}/{key}/{part}'*, *url_format_no_part='{base}/{key}'*, *parts={'permissions': 'auth', 'data': 'data', 'metadata': 'metadata'}*)
Store implementation which gets and sets from a web server

This store expects a server which exposes URLs for each key. By default these URLs are of the form:

```
<base>/<key>/<part>
```

Where <*base*> is a common prefix, <*key*> is the key of interest, and <*part*> is one of "data", "metadata" or "auth". If the store does not follow this format, you can provide a differnt `url_format` argument and a different mapping of <*part*> to aspects of the key.

The server is expected to respond to queries against these URLS in the following ways:

GET <base>/<key>/data return the bytes in the body of the response

PUT <base>/<key>/data accept the data bytes from the body of the request

GET <base>/<key>/metadata return metadata as JSON

PUT <base>/<key>/metadata set the metadata based on JSON contained in the body of the request

POST <base>/<key>/metadata update the metadata based on JSON contained in the body of the request (as *dict.update()*)

GET <base>/<key>/auth return permissions information as JSON

PUT <base>/<key>/auth set the permissions based on JSON contained in the body of the request

POST <base>/<key>/metadata update the permissions based on JSON contained in the body of the request

In addition, a DELETE request to a URL of the form <*base*>/<*key*> should remove the key from the remote store. This pattern is configurable via the `url_format_no_part` argument to the constructor.

In addition, the server should have a query URL which accepts GET reuqests containing a JSON data structure of metadata key, value pairs to filter with, and should return a list of macthing keys, one per line.

connect(*credentials=None*)
Connect to a DynamicURLStore

Parameters **credentials** (*(user_tag, requests.Session)*) – The credentials are a tuple containing ther user's permission tag and a requests Session initialized with appropriate authentication.

delete(*key*)
Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

get(*key*)
Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **value** (instance of Value) - An instance of a Value subclass which holds references to the data, metadata and other information about the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)
Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)
Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_permissions(*key*)
Return the set of permissions the user has

Parameters **key** (*str*) – The key for the resource which you want to know the permissions.

Returns **permissions** (dict of str: set of str) - A dictionary whose keys are the permissions and values are sets of tags which have that permission.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

query(*select=None*, ***kwargs*)

Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys(***kwargs*)

Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)

Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*instance of Value*) – An instance of a Value subclass.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.

- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data(*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_permissions(*key*, *permissions*)

Set the permissions on a key the user owns

Parameters

- **key** (*str*) – The key for the resource which you want to know the permissions.
- **permissions** (*dict of str: set of str*) – A dictionary whose keys are the permissions and values are sets of tags which have that permission. There must be an 'owned' permission with at least one tag.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

transaction(*notes*)
Provide a transaction context manager

This class does not support transactions, so it returns a dummy object.

update_metadata(*key*, *metadata*)
Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

update_permissions(*key*, *permissions*)
Add permissions on a key the user owns

The tags provided in the permissions dictionary will be added to the existing set of tags for each permission.

Parameters

- **key** (*str*) – The key for the resource which you want to know the permissions.
- **permissions** (*dict of str: set of str*) – A dictionary whose keys are the permissions and values are sets of tags which have that permission.

Raises

- **KeyError** - This error will be raised if the key does not exist or the user is not authorized to see it.
- **AuthorizationError** - This error will be raised if user is authorized to see the key, but is not an owner.

Joined Store

class encore.storage.joined_store.**JoinedStore**(*stores*)
A key-value store that joins together several other Key-Value Stores

A joined store is a composite store which takes a list of stores and presents a set of keys that is the union of all the keys that are available in all the stores. When a key is available in multiple stores, then the store which comes first in the list has priority.

All writes are performed into the first store in the list.

Parameters

- **event_manager** – An event_manager which implements the *BaseEventManager* API.
- **stores** (*list of stores*) – The stores that are joined together by this store.

connect (*credentials=None*)

Connect to the key-value store, optionally with authentication

This method creates or connects to any long-lived resources that the store requires.

Parameters **credentials** – An object that can supply appropriate credentials to to authenticate the use of any required resources. The exact form of the credentials is implementation-specific, but may be as simple as a `(username, password)` tuple.

delete (*key*)

Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

disconnect ()

Disconnect from the key-value store

This method disposes or disconnects to any long-lived resources that the store requires.

exists (*key*)

Test whether or not a key exists in the key-value store

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

from_bytes (*key*, *data*, *buffer_size=1048576*)

Efficiently store a bytes object as the data associated with a key.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a bytes object to the underlying data store. The default implementation uses the set() method together with a cStringIO.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

from_file (*key*, *path*, *buffer_size=1048576*)

Efficiently read data from a file into a key in the key-value store.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a path in the filesystem to the underlying data store. The default implementation uses the set() method together with chunked reads from the disk which are fed into the data stream.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

get(*key*)

Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

get_data(*key*)

Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

get_metadata(*key*, *select=None*)

Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

glob(*pattern*)

Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()

Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)
Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)
Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)
Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiset(*keys*, *values*, *buffer_size=1048576*)
Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.

- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)

Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)

Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)
Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

query(*select=None*, ***kwargs*)
Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys(***kwargs*)
Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)
Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data (*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata (*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

to_bytes(*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a bytes object. The default implementation uses the get() method together with chunked reads from the returned data stream and join.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Returns **bytes** - The contents of the file-like object as bytes.

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to extracting the data.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is extracted.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after extracting the data.

to_file(*key*, *path*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a file.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a path in the filesystem. The default implementation uses the get() method together with chunked reads from the returned data stream to the disk.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to disk.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to disk.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to disk.

transaction(*notes*)

Provide a transaction context manager

Implementations which have no native notion of transactions may choose not to implement this.

This method provides a context manager which creates a data store transaction in its __enter__() method, and commits it in its __exit__() method if no errors occur. Intended usage is:

```
with repo.transaction("Writing data..."):
    # everything written in this block is part of the transaction
    ...
```

If the block exits without error, the transaction commits, otherwise the transaction should roll back the state of the underlying data store to the start of the transaction.

Parameters **notes** (*string*) – Some information about the transaction, which may or may not be used by the implementation.

Returns **transaction** (context manager) - A context manager for the transaction.

Events

- **StoreTransactionStartEvent** - This event should be emitted on entry into the transaction.
- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreTransactionEndEvent** - This event should be emitted on successful conclusion of the transaction, before any Set or Delete events are emitted.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set during the transaction.
- **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key for all deleted keys.

update_metadata(*key*, *metadata*)

Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

MountedStore

A store which combines two stores by mounting one of the stores at a particular point in the other store's key space, prefixing all references to keys with the mount point. This is similar in concept to mounting filesystems.

class encore.storage.mounted_store.**MountedStore**(*mount_point*, *mount_store*, *backing_store*)

A key-value store that mounts another store at a particular key prefix

The backing store is treated as read-only, and only modifications are allowed to the first store, and only for keys which match the mounting prefix.

The primary purpose for this is to have a local cache of a subsection of a remote store, such as a StaticURLStore or DynamicURLStore.

Parameters

- **mount_point** (*str*) – Key prefix for the mounted store.
- **mount_store** (AbstractStore) – The store to be mounted
- **backing_store** (AbstractStore) – The store that we are mounting against

connect(*credentials=None*)

Connect to the key-value store, optionally with authentication

This method creates or connects to any long-lived resources that the store requires.

Parameters **credentials** – An object that can supply appropriate credentials to to authenticate the use of any required resources. The exact form of the credentials is implementation-specific, but may be as simple as a `(username, password)` tuple.

delete(*key*)

Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

disconnect()

Disconnect from the key-value store

This method disposes or disconnects to any long-lived resources that the store requires.

get(*key*)

Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store
- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises **KeyError** - If the key is not found in the store, a KeyError is raised.

info()

Get information about the key-value store

Returns **metadata** (dict) - A dictionary of metadata giving information about the key-value store.

is_connected()

Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

push(*key*)
Move a key from the mount store to the backing store

query(*select=None*, ***kwargs*)
Query for keys and metadata matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of metadata for the key-value store.

Parameters

- **select** (*iterable of strings or None*) – An optional list of metadata keys to return. If this is not None, then the metadata dictionaries will only have values for the specified keys populated.
- **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of (key, metadata) tuples where metadata matches all the specified values for the specified metadata keywords. If a key specified in select is not present in the metadata of a particular key, then it will not be present in the returned value.

query_keys(***kwargs*)
Query for keys matching metadata provided as keyword arguments

This provides a very simple querying interface that returns precise matches with the metadata. If no arguments are supplied, the query will return the complete set of keys for the key-value store.

This is equivalent to `self.query(**kwargs).keys()`, but potentially more efficiently implemented.

Parameters **kwargs** – Arguments where the keywords are metadata keys, and values are possible values for that metadata item.

Returns **result** (iterable) - An iterable of key-value store keys whose metadata matches all the specified values for the specified metadata keywords.

set(*key*, *value*, *buffer_size=1048576*)
Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.

- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_data(*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

transaction(*notes*)

Provide a transaction context manager

Implementations which have no native notion of transactions may choose not to implement this.

This method provides a context manager which creates a data store transaction in its __enter__() method, and commits it in its __exit__() method if no errors occur. Intended usage is:

```
with repo.transaction("Writing data..."):
    # everything written in this block is part of the transaction
    ...
```

If the block exits without error, the transaction commits, otherwise the transaction should roll back the state of the underlying data store to the start of the transaction.

Parameters **notes** (*string*) – Some information about the transaction, which may or may not be used by the implementation.

Returns **transaction** (context manager) - A context manager for the transaction.

Events

- **StoreTransactionStartEvent** - This event should be emitted on entry into the transaction.
- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreTransactionEndEvent** - This event should be emitted on successful conclusion of the transaction, before any Set or Delete events are emitted.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set during the transaction.
- **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key for all deleted keys.

update_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Simple Authenticating Store

This module provides a simple wrapper for a store that implements a simple authentication scheme. This may be used as a base for more complex and fine-grained authentication.

By default it authenticates by computing a (salted) hash of the user's password and validates it against the hash stored in an appropriate key. Authenticated users then have full access to all keys.

Subclasses can refine this behaviour by overriding the check_permissions() method to provide different or more controlled permissioning.

encore.storage.simple_auth_store.**make_encoder**(*salt*, *hasher=None*)

Create a moderately secure salted encoder

Parameters

- **salt** (*bytes*) – A salt that is added to the user-supplied password before hashing. This salt should be kept secret, but needs to be remembered across invocations (ie. the same salt needs to be used every time the password is encoded).

- **hasher** (*callable*) – A callable that takes a string and returns a cryptographic hash of the string. The default is *sha1_hasher()*.

encore.storage.simple_auth_store.**sha1_hasher**(*s*)
A simple utility function for producing a sha1 digest of a string.

class encore.storage.simple_auth_store.**SimpleAuthStore**(*store*, *encoder*, *user_key_path='.user_'*, *user_key_store=None*)
A key-value store that wraps another store and implements simple authentication

This wraps an existing store with no notion of authentication and provides simple username/password authentication, storing a hash of the password in the wrapped store.

The base implementation has all-or-nothing

Parameters

- **event_manager** – An event_manager which implements the *BaseEventManager* API.
- **store** (*AbstractStore instance*) – The wrapped store that actually holds the data.
- **encoder** (*callable*) – A callable that computes the password hash.
- **user_key_path** (*str*) – The prefix to put before the username for the keys that store the user's information. At present these keys must simply hold the encoded hash of the user's password.
- **user_key_store** (*AbstractStore instance*) – The store to store the user keys in. Defaults to the wrapped store.

check_permissions(*key=None*)
Return permissions that the user has for the provided key

The default behaviour gives all authenticated users full access to all keys. Subclasses may implement finer-grained controls based on user groups or other permissioning systems.

Parameters **key** (*str or None*) – The key which the permissions are being requested for, or the global permissions if the key is None.

Returns **permissions** (set) - A set of strings chosen from 'connect', 'exists', 'get', 'set', and/or 'delete' which express the permissions that the user has on that particular key.

connect(*credentials=None*)
Connect to the key-value store, optionally with authentication

This method creates or connects to any long-lived resources that the store requires.

Parameters **credentials** – A dictionary with keys 'username' and 'password'.

delete(*key*)
Delete a key from the repsository.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Events **StoreDeleteEvent** - On successful completion of a transaction, a StoreDeleteEvent should be emitted with the key.

Raises **AuthenticationError** - If the user has no rights to delete the key, then an Authentication error is raised.

disconnect()
Disconnect from the key-value store

This method disposes or disconnects to any long-lived resources that the store requires.

exists(*key*)
Test whether or not a key exists in the key-value store

If a user does not have 'exists' permissions for this key, then it will return `False`, even if the key exists in the underlying store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **exists** (bool) - Whether or not the key exists in the key-value store.

from_bytes(*key*, *data*, *buffer_size=1048576*)
Efficiently store a bytes object as the data associated with a key.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a bytes object to the underlying data store. The default implementation uses the set() method together with a cStringIO.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*bytes*) – The data as a bytes object.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

from_file(*key*, *path*, *buffer_size=1048576*)
Efficiently read data from a file into a key in the key-value store.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from a path in the filesystem to the underlying data store. The default implementation uses the set() method together with chunked reads from the disk which are fed into the data stream.

This makes no attempt to set metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to read the data from.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

get(*key*)
Retrieve a stream of data and metdata from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns

- **data** (file-like) - A readable file-like object that provides stream of data from the key-value store

- **metadata** (dictionary) - A dictionary of metadata for the key.

Raises

- **KeyError** - If the key is not found in the store, or does not exist for the user, a KeyError is raised.
- **AuthenticationError** - If the user has no rights to get the key, then an Authentication error is raised.

get_data(*key*)

Retrieve a stream from a given key in the key-value store.

Parameters **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.

Returns **data** (file-like) - A readable file-like object the that provides stream of data from the key-value store.

Raises

- **KeyError** - This will raise a key error if the key is not present in the store.
- **AuthenticationError** - If the user has no rights to get the key, then an Authentication error is raised.

get_metadata(*key*, *select=None*)

Retrieve the metadata for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the result. If unspecified, then return the entire metadata dictionary.

Returns **metadata** (dict) - A dictionary of metadata associated with the key. The dictionary has keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises

- **KeyError** - This will raise a key error if the key is not present in the store.
- **AuthenticationError** - If the user has no rights to get the key, then an Authentication error is raised.

glob(*pattern*)

Return keys which match glob-style patterns

Parameters **pattern** (*string*) – Glob-style pattern to match keys with.

Returns **result** (iterable) - A iterable of keys which match the glob pattern.

is_connected()

Whether or not the store is currently connected

Returns **connected** (bool) - Whether or not the store is currently connected.

multiget(*keys*)

Retrieve the data and metadata for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of (file-like, dict) tuples) - An iterator of (data, metadata) pairs.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_data(*keys*)

Retrieve the data for a collection of keys.

Parameters **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.

Returns **result** (iterator of file-like) - An iterator of file-like data objects corresponding to the keys.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiget_metadata(*keys*, *select=None*)

Retrieve the metadata for a collection of keys in the key-value store.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **select** (*iterable of strings or None*) – Which metadata keys to populate in the results. If unspecified, then return the entire metadata dictionary.

Returns **metadatas** (iterator of dicts) - An iterator of dictionaries of metadata associated with the key. The dictionaries have keys as specified by the select argument. If a key specified in select is not present in the metadata, then it will not be present in the returned value.

Raises **KeyError** - This will raise a key error if the key is not present in the store.

multiset(*keys*, *values*, *buffer_size=1048576*)

Set the data and metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and values have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **values** (*iterable of (file-like, dict) tuples*) – An iterator that provides the (data, metadata) pairs for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.

- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_data(*keys*, *datas*, *buffer_size=1048576*)

Set the data for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and datas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **datas** (*iterable of file-like objects*) – An iterator that provides the data file-like objects for the corresponding keys.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiset_metadata(*keys*, *metadatas*)

Set the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

multiupdate_metadata(*keys*, *metadatas*)

Update the metadata for a collection of keys.

Where supported by an implementation, this should perform the whole collection of sets as a single transaction.

Like zip() if keys and metadatas have different lengths, then any excess values in the longer list should be silently ignored.

Parameters

- **keys** (*iterable of strings*) – The keys for the resources in the key-value store. Each key is a unique identifier for a resource within the key-value store.
- **metadatas** (*iterable of dicts*) – An iterator that provides the metadata dictionaries for the corresponding keys.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata for each key that was set.

set (*key*, *value*, *buffer_size=1048576*)

Store a stream of data into a given key in the key-value store.

This may be left unimplemented by subclasses that represent a read-only key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **value** (*tuple of file-like, dict*) – A pair of objects, the first being a readable file-like object that provides stream of data from the key-value store. The second is a dictionary of metadata for the key.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Raises **AuthenticationError** - If the user has no rights to set the key, then an Authentication error is raised.

set_data (*key*, *data*, *buffer_size=1048576*)

Replace the data for a given key in the key-value store.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **data** (*file-like*) – A readable file-like object the that provides stream of data from the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to the underlying store.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to the underlying store.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to the underlying store.
- **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Raises **AuthenticationError** - If the user has no rights to set the key, then an Authentication error is raised.

set_metadata(*key*, *metadata*)

Set new metadata for a given key in the key-value store.

This replaces the existing metadata set for the key with a new set of metadata.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Raises **AuthenticationError** - If the user has no rights to set the key, then an Authentication error is raised.

to_bytes(*key*, *buffer_size=1048576*)

Efficiently store the data associated with a key into a bytes object.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a bytes object. The default implementation uses the get() method together with chunked reads from the returned data stream and join.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Returns **bytes** - The contents of the file-like object as bytes.

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to extracting the data.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is extracted.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after extracting the data.

to_file(*key*, *path*, *buffer_size=1048576*)
Efficiently store the data associated with a key into a file.

This method can be optionally overriden by subclasses to proved a more efficient way of copy the data from the underlying data store to a path in the filesystem. The default implementation uses the get() method together with chunked reads from the returned data stream to the disk.

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **path** (*string*) – A file system path to store the data to.
- **buffer_size** (*int*) – An optional indicator of the number of bytes to read at a time. Implementations are free to ignore this hint or use a different default if they need to. The default is 1048576 bytes (1 MiB).

Events

- **StoreProgressStartEvent** - For buffering implementations, this event should be emitted prior to writing any data to disk.
- **StoreProgressStepEvent** - For buffering implementations, this event should be emitted periodically as data is written to disk.
- **StoreProgressEndEvent** - For buffering implementations, this event should be emitted after finishing writing to disk.

update_metadata(*key*, *metadata*)
Update the metadata for a given key in the key-value store.

This performs a dictionary update on the existing metadata with the provided metadata keys and values

Parameters

- **key** (*string*) – The key for the resource in the key-value store. They key is a unique identifier for the resource within the key-value store.
- **metadata** (*dict*) – A dictionary of metadata to associate with the key. The dictionary keys should be strings which are valid Python identifiers.

Events **StoreSetEvent** - On successful completion of a transaction, a StoreSetEvent should be emitted with the key & metadata

Raises **AuthenticationError** - If the user has no rights to set the key, then an Authentication error is raised.

3.2.2 Indices and tables

- genindex
- modindex
- search

3.2.3 License

This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

Unless otherwise noted:

3.2.4 util.human_date module

3.2.5 concurrent.futures.enhanced_thread_pool_executor

3.3 Concurrent Package

The `encore.concurrent` module provides utilities and libraries to assist with threaded and other parallel code.

The `encore.concurrent.futures` subpackage provides an enhanced version of the `concurrent.futures` package for Python 2.7 with some useful experimental additions.

The *`encore.concurrent.threadtools`* module provides some utilities that encapsulate useful patterns in threaded code.

3.3.1 Contents

encore.concurrent Package

threadtools Module

Module of useful routines for working with concurrency.

encore.concurrent.threadtools.**synchronized**(*func*)
Decorator that prevents simultaneous execution of a function

This decorator that ensures that only one thread at a time can be executing the decorated function at the same time by using a dedicated anonymous lock.

encore.concurrent.futures Package

ThreadPool Executors

enhanced_thread_pool_executor Module

synchronous Module

future Module

abc_work_scheduler Module

asynchronizer Module

serializer Module

serializing_asynchronizer Module

3.3.2 Indices and tables

- genindex
- modindex
- search

3.3.3 License

This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

Unless otherwise noted:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Enthought, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

3.3.4 util.human_date module

Copyright 2009 Jai Vikram Singh Verma (jaivikram[dot]verma[at]gmail[dot]com) Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

3.3.5 concurrent.futures.enhanced_thread_pool_executor

Copyright 2009 Brian Quinlan. All Rights Reserved. Licensed to PSF under a Contributor Agreement.

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

License

This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative.

Unless otherwise noted:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Enthought, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

CHAPTER 6

util.human_date module

CHAPTER 7

concurrent.futures.enhanced_thread_pool_executor

e

Symbols

A

B

C

D

E

F

G

I

J

K

M

O

P

Q

R

S

T

U

V