Sunspear Documentation

Release 0.1a1

Numan Sachwani

Jan 03, 2018

Contents

Sunspear in a library for storing activity streams. It is an implementation of the JSON Activity Stream 1.0 specification. Currently, Sunspear only supports `Riak` as a storage backend, however, it is modular enough to allow the use of custom backend.

Contents:

CHAPTER 1

User Guide

1.1 Introduction

Sunspear is a python library which provides an API to store and manage activity stream items.

1.1.1 What it is

Sunspear is a library which allows you to add feeds and activity streams to your applications. It serializes these activities and feed items as described in the JSON Activity Stream 1.0 specifications.

The specifications look scary, dull and boring (and they are), but essentially, an activity for:

```
``John Doe`` added ``Jane Doe`` to the group ``Party Planning``
```

boils down to:

```
{
    "actor": {
        "objectType": "user",
        "displayName": "John Doe",
        "id": "user:1234", # Totally optional, it will be generated for you if you don
↪'t provide one.
        "title": "Manager",
        "foo": "bar"
    },
    "object": {
        "objectType": "user",
        "displayName": "Jane Doe",
        "title": "Devloper",
        "foo": "baz"
    },
    "target": {
        "objectType": "group",
        "displayName": "Party Planning"
    },
    "bar": "bee",
    "verb": "added"
}
```

The main takeaway points are:

- An `activity` is composed of `objects`.
- An `activity` must define an `actor`, `verb` and `object`.
- An `object` must define an `objectType` and a `displayName`
- `objects` and `activities` may contain arbitrary attributes.

Note: For more info, see the specifications for activity and object.

Sunspear also implements parts of some extensions to the specificiations. More specifically, Audience Targeting and Responses.

1.1.2 What it isn't

Sunspear strictly deals with storage and retrival of JSON activity stream items. It does not include all adquate indexes that allow you to build a fully fledged feed system.

For indexing, you'll probably want to use something like Sandsnake, a sorted index backed by redis.

1.2 Quick Start Guide

1.2.1 Usage

Sunspear provides a very simple api managing activity stream items

Initialize

You initialize sunspear by providing `sunspear.clients.SunspearClient` with an instance of a backend. All backends are located in `sunspear.backends` and extend the `sunspear.backend.base.BaseBackend`:

```
from sunspear.backends.RiakBackend import RiakBackend
from sunspear.clients import SunspearClient

import datetime

client = SunspearClient(RiakBackend(**{
    "host_list": [{'port': 8087}],
    "defaults": {'host': '127.0.0.1'},
}))
```

Create Objects

Once you have a reference to the `SunspearClient`, you can, create objects:

```
obj = client.create_object({
    "objectType": "user"
    "displayName": "John Doe",
    "email": "jdoe@gmail.com",
    "id": "user:1234",
})
```

Note: If you do not specify the `id` of an object, one will be automatically generated for you. This also applies for the `published` date.

Create Activity

You can also create an `activity`. As per the JSON Activity Stream 1.0 specifications, an activity must have a `verb`, `actor` and an `object`.

Creating an activity for:

```
John Doe created the team "Marketing"
```

may look something like this:

```
activity = client.create_activity({
    "verb": "create",
    "actor": "user:1234",
    "object": {
        "objectType": "team",
        "displayName": "Marketing",
    },
})
```

Couple of intresting things to note here:

1. We used the `id` for the `actor` instead of the full blown `object` because we created the `object` earlier.
2. We didn't specify an `id` for our `team` object. One will be automatically generated.
3. The verb is arbratry. It can be anything except for response and activity summary verbs. For list a list of common `verbs` and `objects` you may want to use, see the Activity Base Schema.
4. `create_activity` returns the fully parsed activity.

Note: The reason you can't use response and activity summary verbs is because **Sunspear** uses some of them internally.

Create Responses

You can create responses to activities such as liking an activity or replying to an activity. Sunspear supports a few of the response types described here.

All methods that create responses, return the newly created response activity and the original activity the response was created for with the response activity embedded.

Note: **Sunspear** does responses a little bit differently than what is describe in the specifications.

Responses themselves are fully fledged `activities` and not `objects`. This was untimatly done to provide maximum flexibility.

Create a Reply

You can create replies to activities.

```
reply_activity, original_activity = client.create_reply(activity['id'], "user:1234",
    "This is my Reply!")
```

Create a Like

You can like activities.

```
like_activity, original_activity = client.create_like(activity['id'], "user:1234")
```

Delete Reply

```
original_activity = client.delete_reply(reply_activity["id"])
```

Delete Like

```
original_activity = client.delete_like(reply_activity["id"])
```

Get Activities

You can get activities by providing a list of `ids`.

```
client.get_activities([activity["id"], "1234", "3456"])
```

Note: If the activity with the id does not exist, it is simply ignored.

Get Objects

You can get objects by providing a list of `ids`.

```
client.get_objects([obj["id"], "1234", "3456"])
```

Note: If the object with the id does not exist, it is simply ignored.

CHAPTER 2

Developer Guide

CHAPTER 3

API Guide

3.1 Sunspear Client

3.1.1 sunspear.clients

class `sunspear.clients.`**`SunspearClient`**(*backend*, ***kwargs*)

The class is used to create, delete, remove and update activity stream items. This is the main class you use to interact with `sunspear`

`clear_all`()

Deletes all activity stream data

`clear_all_activities`()

Deletes all activities data

`clear_all_objects`()

Deletes all objects data

`create_activity`(*actstream_dict*)

Creates an activity. You can provide objects for activities as dictionaries or as ids for already existing objects.

If you provide a dictionary for an object, it is saved as a new object. If you provide an object id and the object does not exist, it is saved anyway, and returned as an empty dictionary when retriving the activity.

Parameters **`actstream_dict`** (*`dict`*) – a dictionary representing the `activity` we want to store in the backend.

`create_like`(*activity*, *actor*, *content=”*, *extra={}*, ***kwargs*)

Creates a `like` for an activity.

Parameters

- **`activity`** (*`a string or dict`*) – the activity we want to create the sub-item for
- **`actor`** (*`a string or dict`*) – the `object` creating the sub-activity

- **content** (*a string or dict*) – a string or an `object` representing the content of the sub-activity
- **extra** (*dict*) – additional data the is to be included as part of the `sub-activity` activity

create_object (*object_dict*)
Creates an object that can be used as part of an activity. If you specific and object with an id that already exists, that object is overidden.

Parameters **object_dict** (*dict*) – a dictionary representing the object we want to store in the backend.

create_reply (*activity*, *actor*, *content*, *extra={}*, ***kwargs*)
Creates a `reply` for an activity.

Parameters

- **activity** (*a string or dict*) – the activity we want to create the sub-item for
- **actor** (*a string or dict*) – the `object` creating the sub-activity
- **content** (*a string or dict*) – a string or an `object` representing the content of the sub-activity
- **extra** (*dict*) – additional data the is to be included as part of the `sub-activity` activity

delete_activity (*activity_id*, ***kwargs*)
Deletes an activity item and all associated sub items

Parameters **activity_id** (*string*) – The id of the activity we want to create a reply for

delete_like (*activity_id*, ***kwargs*)
Deletes a `like` made on an activity. This will also update the corresponding activity.

Parameters **activity_id** (*string*) – the id of the like activity to delete.

delete_reply (*activity_id*, ***kwargs*)
Deletes a `reply` made on an activity. This will also update the corresponding activity.

Parameters **activity_id** (*string*) – the id of the reply activity to delete.

get_activities (*activity_ids=[]*, ***kwargs*)
Gets a list of activities. Specific backends may support other arguments. Please see reference of the specific backends to see all `kwargs` supported.

Parameters **activity_ids** (*list*) – The list of activities you want to retrieve

get_backend ()
The backend the client was initialized with.

Returns reference to the backend the client was initialized with.

get_objects (*object_ids=[]*)
Gets a list of objects by object_ids.

Parameters **object_ids** (*list*) – a list of objects

3.2 Sunspear Client

3.2.1 sunspear.backends.base

class `sunspear.backends.base.`**`BaseBackend`**

`activity_create`(*activity*, ***kwargs*)
Stores a new activity to the backend.

Parameters **`activity`** (*`dict`*) – a dict representing the activity

Returns a dict representing the newly stored activity

`activity_delete`(*activity*, ***kwargs*)
Performs the task of actually deleting the activity from the backend.

Parameters **`activity`** (*`dict`*) – a dict representing the activity

`activity_exists`(*activity*, ***kwargs*)
Determins if an `activity` already exists in the backend.

Parameters **`activity`** (*`dict`*) – the activity we want to determin if it exists

Returns `True` if the `activity` exists, otherwise `False`

`activity_get`(*activity*, ***kwargs*)

`activity_update`(*activity*, ***kwargs*)
Performs the actual task of updating the activity in the backend.

Parameters **`activity`** (*`dict`*) – a dict representing the activity

Returns a dict representing the newly stored activity

`clear_all_activities`()
Clears all activities from the backend.

`clear_all_objects`()
Clears all objects from the backend.

`create_activity`(*activity*, ***kwargs*)
Stores a new `activity` in the backend. If an object with the same id already exists in the backend, a `SunspearDuplicateEntryException` is raised. If an ID is not provided, one is generated on the fly.

Activities that provide `objects` as dictionaries have their objects processed and stored using `create_obj`, and the `objects` are replaced with their id's within the activity.

Parameters **`activity`** (*`dict`*) – activity we want to store in the backend

Raises `SunspearDuplicateEntryException` if the record already exists in the database.

Returns dict representing the new activity.

`create_obj`(*obj*, ***kwargs*)
Stores a new `obj` in the backend. If an object with the same id already exists in the backend, a `SunspearDuplicateEntryException` is raised. If an ID is not provided, one is generated on the fly.

Parameters **`obj`** (*`dict`*) – obj we want to store in the backend

Raises `SunspearDuplicateEntryException` if the record already exists in the database.

Returns dict representing the new obj.

`create_sub_activity`(*activity*, *actor*, *content*, *extra={}*, *sub_activity_verb=”*, ***kwargs*)
Creates a new sub-activity as a child of `activity`.

Parameters

- **`activity`** (*`a string or dict`*) – the activity we want to create the sub-item for
- **`actor`** (*`a string or dict`*) – the `object` creating the sub-activity
- **`content`** (*`a string or dict`*) – a string or an `object` representing the content of the sub-activity
- **`extra`** (*`dict`*) – additional data the is to be included as part of the `sub-activity` activity
- **`sub_activity_verb`** (*`string`*) – the verb of the sub activity

Returns a tuple containing the new sub activity and the original activity the sub activity was created for.

`dehydrate_activities`(*activities*)
Takes a raw list of activities returned from riak and replace keys with contain ids for riak objects with actual riak object

`delete_activity`(*activity*, ***kwargs*)
Deletes an existing activity from the backend.

Parameters **`activity`** (*`dict`*) – a dict representing the activity

Raises `SunspearInvalidActivityException` if the activity doesn’t have a valid id.

`delete_obj`(*obj*, ***kwargs*)
Deletes an existing obj from the backend.

raises:

- `SunspearInvalidObjectException` – if the obj doesn’t have a valid id.

Parameters **`obj`** (*`dict`*) – a dict representing the obj

Raises SunspearInvalidObjectException

`delete_sub_activity`(*sub_activity*, *sub_activity_verb*, ***kwargs*)
Deletes a `sub_activity` made on an activity. This will also update the corresponding activity.

Parameters

- **`sub_activity`** (*`string`*) – the id of the reply activity to delete
- **`sub_activity_verb`** (*`string`*) – the verb of the sub activity

`get_activity`(*activity_ids=[]*, ***kwargs*)
Gets an activity or a list of activities from the backend.

Parameters **`activity_ids`** (*`list`*) – a list of ids of activities that will be retrieved from the backend.

Returns a list of activities. If an activity is not found, a partial list should be returned.

`get_new_id`()
Generates a new unique ID. The default implementation uses uuid1 to generate a unique ID.

Returns a new id

get_obj (*obj_ids=[]*, ***kwargs*)
Gets an obj or a list of activities from the backend.

Parameters **obj** (`list`) – a list of ids of activities that will be retrieved from the backend.

Returns a list of activities. If an obj is not found, a partial list should be returned.

get_sub_activity_attribute (*sub_activity_verb*)

get_sub_activity_model (*sub_activity_verb*)

is_sub_activity_verb_valid (*sub_activity_verb*)

obj_create (*obj*, ***kwargs*)
Stores a new obj to the backend.

Parameters **obj** (`dict`) – a dict representing the obj

Returns a dict representing the newly stored obj

obj_delete (*obj*, ***kwargs*)

obj_exists (*obj*, ***kwargs*)
Determins if an `object` already exists in the backend.

Parameters **obj** (`dict`) – the activity we want to determin if it exists

Returns `True` if the `object` exists, otherwise `False`

obj_get (*obj*, ***kwargs*)

obj_update (*obj*, ***kwargs*)

sub_activity_create (*activity*, *actor*, *content*, *extra={}*, *sub_activity_verb=''*, *sub_activity_attribute=''*, ***kwargs*)
Creates a new sub-activity as a child of `activity`.

Parameters

- **activity** (`a string or dict`) – the activity we want to create the sub-item for
- **actor** (`a string or dict`) – the `object` creating the sub-activity
- **content** (`a string or dict`) – a string or an `object` representing the content of the sub-activity
- **extra** (`dict`) – additional data the is to be included as part of the `sub-activity` activity
- **sub_activity_verb** (`string`) – the verb of the sub activity
- **sub_activity_attribute** (`string`) – the attribute in the activity the `sub-activity` will be a part of

Returns a tuple containing the new sub activity and the original activity the sub activity was created for.

sub_activity_delete (*sub_activity*, *sub_activity_verb*, ***kwargs*)

update_activity (*activity*, ***kwargs*)
Updates an existing activity in the backend. If the object does not exist, it is created in the backend.

Parameters **activity** (`dict`) – a dict representing the activity

Raises `SunspearInvalidActivityException` if the activity doesn't have a valid id.

Returns a dict representing the newly stored activity

update_obj(*obj*, ***kwargs*)
Updates an existing obj in the backend. If the object does not exist, it is created in the backend.

raises:

- `SunspearInvalidObjectException` – if the obj doesn't have a valid id.

Parameters **obj** (*dict*) – a dict representing the obj

Raises SunspearInvalidObjectException

Returns a dict representing the newly stored obj

3.2.2 sunspear.backends.riak

class sunspear.backends.riak.**RiakBackend**(*protocol='pbc'*, *nodes=[]*, *objects_bucket_name='objects'*, *activities_bucket_name='activities'*, ***kwargs*)

activity_create(*activity*, ***kwargs*)
Creates an activity. You can provide objects for activities as dictionaries or as ids for already existing objects.

If you provide a dictionary for an object, it is saved as a new object.

If you provide an object id and the object does not exist, it is saved anyway, and returned as an empty dictionary when retriving the activity later.

activity_delete(*activity*, ***kwargs*)
Deletes an activity item and all associated sub items

activity_exists(*activity*, ***kwargs*)

activity_get(*activity_ids=[]*, *raw_filter=''*, *filters={}*, *include_public=False*, *audience_targeting={}*, *aggregation_pipeline=[]*, ***kwargs*)
Gets a list of activities. You can also group activities by providing a list of attributes to group by.

Parameters

- **activity_ids** (*list*) – The list of activities you want to retrieve
- **filters** (*dict*) – filters list of activities by key, value pair. For example, `{'verb': 'comment'}` would only return activities where the `verb` was `comment`. Filters do not work for nested dictionaries.
- **raw_filter** (*string*) – allows you to specify a javascript function as a string. The function should return `true` if the activity should be included in the result set or `false` it shouldn't. If you specify a raw filter, the filters specified in `filters` will not run. How ever, the results will still be filtered based on the `audience_targeting` parameter.

- **include_public** (*boolean*) – If `True`, and the `audience_targeting` dictionary is defined, activities that are not targeted towards anyone are included in the results
- **audience_targeting** (*dict*) – Filters the list of activities targeted towards a particular audience. The key for the dictionary is one of `to`, `cc`, `bto`, or `bcc`. The values are an array of object ids
- **aggregation_pipeline** (array of `sunspear.aggregators.base.BaseAggregator`) – modify the final list of activities. Exact results depends on the implementation of the aggregation pipeline

Returns list – a list of activities matching `activity_ids`. If the activities is not found, it is not included in the result set. Activities are returned in the order of ids provided.

activity_update(*activity*, ***kwargs*)

clear_all(***kwargs*)
Deletes all activity stream data from riak

clear_all_activities(***kwargs*)
Deletes all activities data from riak

clear_all_objects(***kwargs*)
Deletes all objects data from riak

get_new_id()
Generates a new unique ID. The default implementation uses uuid1 to generate a unique ID.

Returns a new id

obj_create(*obj*, ***kwargs*)

obj_delete(*obj*, ***kwargs*)

obj_exists(*obj*, ***kwargs*)

obj_get(*obj*, ***kwargs*)
Given a list of object ids, returns a list of objects

obj_update(*obj*, ***kwargs*)

set_activity_indexes(*riak_object*)
Store indexes specific to an `Activity`. Stores the following indexes: 1. `verb` of the `Activity` 2. `actor` of the `Activity` 3. `object` of the `Activity` 4. if target is defined, verb for the `target` of the Activity

Parameters **riak_object** (`RiakObject`) – a RiakObject representing the model of the class

set_general_indexes(*riak_object*)
Sets the default riak 2Is.

Parameters **riak_object** (`RiakObject`) – a RiakObject representing the model of the class

set_sub_item_indexes(*riak_object*, ***kwargs*)
Store indexes specific to a sub-activity. Stores the following indexes: 1. id of the the parent `Activity` of this sub-activity

Parameters **riak_object** (`RiakObject`) – a RiakObject representing the model of the class

sub_activity_create(*activity*, *actor*, *content*, *extra={}*, *sub_activity_verb=”*, *published=None*, ***kwargs*)

`sub_activity_delete` (*sub_activity*, *sub_activity_verb*, ***kwargs*)

Deletes a sub_activity made on an activity. This will also update the corresponding parent activity.

Parameters

- **`sub_activity`** (*`string`*) – the id of the reply activity to delete.
- **`sub_activity_verb`** (*`string`*) – the verb of the sub activity

Returns a dict representing the updated parent activity

CHAPTER 4

References

4.1 License

Sunspear is licensed under the Apache 2 license.

The full license text can be found below (*Sunspear License*).

4.1.1 Authors

Sentry was originally written and is maintained by Numan Sachwani.

A lit of additional contributors can be seen on GitHub.

4.1.2 Sunspear License

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file,

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index