python-simple-workflow Documentation

Release 0.1a

Oleiade

May 27, 2013

CONTENTS

Under heavy development, and not yet usable

python-simple-workflow is a wrapper for Amazon Simple Workflow service. It aims to provide some abstractions over Boto library SWF API implementation, like querysets and objects over commonly used concepts: `Domains`, `Workflows`, `Activities`, and so on.

It is under MIT license, and any ideas, features requests, patches, pull requests to improve it are of course welcome.

CHAPTER

ONE

QUICKSTART

Still under heavy development

python-simple-workflow is a wrapper for Amazon Simple Workflow service. It aims to provide some abstractions over Boto library SWF API implementation, like querysets and objects over commonly used concepts: `Domains`, `Workflows`, `Activities`, and so on.

1.1 Installation

```
pip install simple-workflow
```

1.2 Authentication

To be able to communicate with Amazon service, the python simple workflow package modules have to be aware of your AWS credentials. Three credentials providing methods are available and evaluated by the module in the following order

Credential file a `.swf` is seeked in the user's home directory, it's content has to match the following pattern

```
[credentials]
aws_access_key_id=<aws_access_key_id>
aws_secret_access_key=<aws_secret_access_key>
```

Environment variables python-simple-workflow will check for `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables to be set to relevant values.

Helper function if neither previous methods were used, you'll still be able to set the global module aws credentials using the `swf.credentials.set_aws_credentials` method.

```
>>> from swf.credentials import set_aws_credentials
>>> set_aws_credentials('MYAWSACCESSKEYID', 'MYAWSSECRETACCESSKEY')

# And then you're good to go...
>>> queryset = DomainQuery()
>>> queryset.all()
[Domain('test1'), Domain('test2')]
```

1.3 Batteries Included

1.3.1 Models

Simple Workflow entities such as domains, workflow types, workflow executions and activity types are to be manipulated through python-simple-workflow using `models`. They are immutable `swf` objects representations providing an interface to objects attributes, local/remote objects synchronization and changes watch between these local and remote objects.

```
# Models resides in swf.models module
>>> from swf.models import Domain, WorkflowType, WorkflowExecution, ActivityType

# Once imported you're ready to create a local model instance
>>> D = Domain(
    "my-test-domain-name",
    description="my-test-domain-description",
    retention_period=60
)

# a Domain model local instance has been created, but nothing has been
# sent to amazon. To do so, you have to save it.
>>> D.save()
```

Now you have a local `Domain` model object, and if no errors were raised, the `save` method have saved amazon-side. But, sometimes, you won't be able to know if the model you're manipulating has an upstream version: whether you've acquired it through a queryset, or the remote object has been deleted for example. Fortunately, models are shipped with a set of functions to make sure your local objects keep synced and consistent.

```
# Exists method let's you know if you're model instance has an upstream version
>>> D.exists
True

# What if changes have been made to the remote object?
# synced  and changes methods help ensuring local and remote models
#are still synced and which changes have been maid.
>>> D.is_synced
True
>>> D.changes
ModelDiff()
```

What if your local object is out of sync? Models `upstream` method will fetch the remote version of your object and will build a new model instance using it's attributes.

```
>>> D.is_synced
False
>>> D.changes
ModelDiff(
    Difference('status', 'REGISTERED', 'DEPRECATED')
)

# Let's pull the upstream version
>>> D = D.upstream()
>>> D.is_synced
True
>>> D.changes
ModelDiff()
```

1.3.2 QuerySets

Models can be retrieved and instantiated via querysets. To continue over the django comparison, they're behaving like django managers.

```
# As querying for models needs a valid connection to amazon service,
# Queryset objects cannot act as classmethods proxy and have to be instantiated;
# most of the time against a Domain model instance
>>> from swf.querysets import DomainQuerySet, WorkflowTypeQuerySet

# Domain querysets can be instantiated directly
>>> domain_qs = DomainQuerySet()
>>> workflow_domain = domain_qs.get("MyTestDomain")  # and specific model retieved via .get method
>>> workflow_qs = WorkflowTypeQuerySet(workflow_domain)  # queryset built against model instance exa

>>> workflow_qs.all()
[WorkflowType("TestType1"), WorkflowType("TestType2"),]

>>> workflow_qs.filter(status=DEPRECATED)
[WorkflowType("DeprecatedType1"),]
```

1.3.3 Events

(coming soon)

1.3.4 History

(coming soon)

1.3.5 Decisions

(coming soon)

1.3.6 Actors

Swf workflows are based on a worker-decider pattern. Every actions in the flow is executed by a worker which runs supplied activity tasks. And every actions is the result of a decision taken by the decider reading the workflow events history and deciding what to do next. In order to ease the development of such workers and decider, python-simple-workflow exposes base classes for them located in `swf.actors` submodule.

- An `Actor` must basically implement a `start` and `stop` method and can actually inherits from whatever runtime implementation you need: thread, gevent, multiprocess...

```
class Actor(ConnectedSWFObject):
    def __init__(self, domain, task_list)
    def start(self):
    def stop(self):
```

- `Decider` base class implements the core functionality of a swf decider: polling for decisions tasks, and sending back a decision task copleted decision. Every other special needs implementations are left up to the user.

```
class Decider(Actor):
    def __init__(self, domain, task_list)
    def complete(self, task_token, decisions=None, execution_context=None)
    def poll(self, task_list=None, identity=None, maximum_page_size=None)
```

- `Worker` base class implements the core functionality of a swf worker whoes role is to process activity tasks. It is basically able to poll for new activity tasks to process, send back a heartbeat to swf service in order to let it know it hasn't failed or crashed, and to complete, fail or cancel the activity task it's processing.

```
class ActivityWorker(Actor):
    def __init__(self, domain, task_list)
    def cancel(self, task_token, details=None)
    def complete(self, task_token, result=None)
    def fail(self, task_token, details=None, reason=None)
    def heartbeat(self, task_token, details=None)
    def poll(self, task_list=None, **kwargs)
```

CHAPTER

TWO

API

2.1 Models

2.1.1 Domain

2.1.2 Workflow Type

class swf.models.workflow.**WorkflowType**(**args*, ***kw*)

changes
Returns changes between current model instance, and remote object representation

Returns A list of swf.models.base.ModelDiff namedtuple describing differences

Return type list

child_policy
descr.__get__(obj[, type]) -> value

creation_date
descr.__get__(obj[, type]) -> value

decision_tasks_timeout
descr.__get__(obj[, type]) -> value

delete()
Deprecates the workflow type amazon-side

deprecation_date
descr.__get__(obj[, type]) -> value

description
descr.__get__(obj[, type]) -> value

domain
descr.__get__(obj[, type]) -> value

execution_timeout
descr.__get__(obj[, type]) -> value

exists
descr.__get__(obj[, type]) -> value

is_synced
Checks if current Model instance has changes, comparing with remote object representation

Return type bool

name
descr.__get__(obj[, type]) -> value

save()
Creates the workflow type amazon side

start_execution(*workflow_id=None*, *task_list=None*, *child_policy=None*, *execution_timeout=None*, *input=None*, *tag_list=None*, *decision_tasks_timeout=None*)
Starts a Workflow execution of current workflow type

Parameters

- **workflow_id** (*String*) – The user defined identifier associated with the workflow execution
- **task_list** (*String*) – task list to use for scheduling decision tasks for execution of this workflow
- **child_policy** (*CHILD_POLICIES.{TERMINATE | REQUEST_CANCEL | ABANDON}*) – policy to use for the child workflow executions of this workflow execution.
- **execution_timeout** (*String*) – maximum duration for the workflow execution
- **input** (*String*) – Input of the workflow execution
- **tag_list** (*String*) – Tags associated with the workflow execution
- **decision_tasks_timeout** (*String*) – maximum duration of decision tasks for this workflow execution

status
descr.__get__(obj[, type]) -> value

task_list
descr.__get__(obj[, type]) -> value

version
descr.__get__(obj[, type]) -> value

2.1.3 Workflow Execution

class swf.models.workflow.**WorkflowExecution**(**args*, ***kw*)

changes
Returns changes between current model instance, and remote object representation

Returns A list of swf.models.base.ModelDiff namedtuple describing differences

Return type list

child_policy
descr.__get__(obj[, type]) -> value

decision_tasks_timeout
descr.__get__(obj[, type]) -> value

delete()
Deprecates the connected swf object amazon side

domain
descr.__get__(obj[, type]) -> value

execution_timeout
descr.__get__(obj[, type]) -> value

exists
descr.__get__(obj[, type]) -> value

history(**args*, ***kwargs*)
Returns workflow execution history report

Returns The workflow execution complete events history

Return type swf.models.event.History

input
descr.__get__(obj[, type]) -> value

is_synced
Checks if current Model instance has changes, comparing with remote object representation

Return type bool

run_id
descr.__get__(obj[, type]) -> value

save()
Creates the connected swf object amazon side

signal(*signal_name*, *input=None*, **args*, ***kwargs*)
Records a signal event in the workflow execution history and creates a decision task.

The signal event is recorded with the specified user defined `signal_name` and `input` (if provided).

Parameters

- **signal_name** (*str*) – The name of the signal. This name must be meaningful to the target workflow.
- **input** (*str*) – Data to attach to the WorkflowExecutionSignaled event in the target workflow execution's history.

status
descr.__get__(obj[, type]) -> value

tag_list
descr.__get__(obj[, type]) -> value

task_list
descr.__get__(obj[, type]) -> value

upstream()
Instantiates a new upstream version of the model

workflow_id
descr.__get__(obj[, type]) -> value

workflow_type
descr.__get__(obj[, type]) -> value

2.1.4 Event

class `swf.models.event.`**Event**(*id*, *state*, *timestamp*, *raw_data*)
Simple workflow execution event wrapper base class

Intends to be used as a base abstraction for the multiple amazon swf events implementation. Generally events implementation will be instantiated through the `swf.models.event.factory.EventFactory` factory.

It provides basic common attributes, such as the event type, name, json representation key to extract relevant data from, and sets the event id, state and timestamp from the constructor.

Event base class is used in this project to implement `swf.models.event.task.DecisionTaskEvent`, which a typical instance would for example have type 'DecisionTask', name 'DecisionTaskScheduleFailed', id '1' and state 'failed'.

Parameters

- **id** (*string*) – event id provided by amazon service
- **state** (*string*) – event current state
- **timestamp** (*float*) – event creation timestamp
- **raw_data** (*dict*) – raw_event representation provided by amazon service

process_attributes()
Processes the event raw_data attributes_key elements and sets current instance attributes accordingly

2.1.5 Execution History

2.1.6 Activity Type

class swf.models.activity.**ActivityType**(**args*, ***kw*)

changes
Returns changes between current model instance, and remote object representation

Returns A list of swf.models.base.ModelDiff namedtuple describing differences

Return type list

creation_date
descr.__get__(obj[, type]) -> value

delete()
Deprecates the domain amazon side

deprecation_date
descr.__get__(obj[, type]) -> value

description
descr.__get__(obj[, type]) -> value

domain
descr.__get__(obj[, type]) -> value

exists
descr.__get__(obj[, type]) -> value

is_synced
Checks if current Model instance has changes, comparing with remote object representation

Return type bool

name
descr.__get__(obj[, type]) -> value

save()
Creates the activity type amazon side

status
descr.__get__(obj[, type]) -> value

task_heartbeat_timeout
descr.__get__(obj[, type]) -> value

task_list
descr.__get__(obj[, type]) -> value

task_schedule_to_close_timeout
descr.__get__(obj[, type]) -> value

task_schedule_to_start_timeout
descr.__get__(obj[, type]) -> value

task_start_to_close_timeout
descr.__get__(obj[, type]) -> value

version
descr.__get__(obj[, type]) -> value

2.1.7 Activity Task

class swf.models.activity.**ActivityTask**(**args*, ***kw*)

activity_id
descr.__get__(obj[, type]) -> value

activity_type
descr.__get__(obj[, type]) -> value

changes
Returns changes between current model instance, and remote object representation

Returns A list of swf.models.base.ModelDiff namedtuple describing differences

Return type list

delete()
Deprecates the connected swf object amazon side

domain
descr.__get__(obj[, type]) -> value

exists
Checks if the connected swf object exists amazon-side

input
descr.__get__(obj[, type]) -> value

is_synced
Checks if current Model instance has changes, comparing with remote object representation

Return type bool

save()
Creates the connected swf object amazon side

started_event_id
descr.__get__(obj[, type]) -> value

task_list
descr.__get__(obj[, type]) -> value

task_token
descr.__get__(obj[, type]) -> value

upstream()
Instantiates a new upstream version of the model

workflow_execution
descr.__get__(obj[, type]) -> value

2.2 Querysets

2.2.1 DomainQuerySet

class swf.querysets.domain.**DomainQuerySet**(**args*, ***kwargs*)
Swf domain queryset object

Allows the user to interact with amazon's swf domains through a django-queryset like interface

all(*registration_status='REGISTERED'*)
Retrieves every domains

Parameters **registration_status** (*string*) – domain registration status to match, Valid values are: * swf.constants.REGISTERED * swf.constants.DEPRECATED

A typical Amazon response looks like:

```
{
    "domainInfos": [
        {
            "name": "Crawl"
            "status": "REGISTERED",
            "description": "",
        },
    ]
}
```

create(*name*, *status='REGISTERED'*, *description=None*, *retention_period=30*, **args*, ***kwargs*)
Creates a new remote domain and returns the Domain model instance

Parameters

- **name** (*string*) – Name of the domain to register (unique)
- **retention_period** (*Integer*) – Domain's workflow executions records retention in days
- **status** (*string*) – Specifies the registration status of the workflow types to list. Valid values are: * swf.constants.REGISTERED * swf.constants.DEPRECATED
- **description** (*string*) – Textual description of the domain

get(*name*)
Fetches the Domain with *name*

Parameters **name** (*string*) – name of the domain to fetch

A typical Amazon response looks like:

```
{
    "configuration": {
        "workflowExecutionRetentionPeriodInDays": "7",
    },
    "domainInfo": {
        "status": "REGISTERED",
        "name": "CrawlTest",
    }
}
```

get_or_create(*name*, *status='REGISTERED'*, *description=None*, *retention_period=30*, **args*, ***kwargs*)

Fetches, or creates the Domain with *name*

When fetching trying to fetch a matching domain, only name parameter is taken in account. Anyway, If you'd wanna make sure that in case the domain has to be created it is made with specific values, just provide it.

Parameters

- **name** (*string*) – name of the domain to fetch or create
- **retention_period** (*Integer*) – Domain's workflow executions records retention in days
- **status** (*string*) – Specifies the registration status of the workflow types to list. Valid values are: * `swf.constants.REGISTERED` * `swf.constants.DEPRECATED`
- **description** (*string*) – Textual description of the domain

Returns Fetched or created Domain model object

Return type Domain

2.2.2 WorkflowTypeQuerySet

class `swf.querysets.workflow.`**WorkflowTypeQuerySet**(*domain*, **args*, ***kwargs*)

all(*registration_status='REGISTERED'*)

Retrieves every Workflow types

Parameters **registration_status** (*string*) – workflow type registration status to match, Valid values are: * `swf.constants.REGISTERED` * `swf.constants.DEPRECATED`

A typical Amazon response looks like:

```
{
    "typeInfos": [
        {
            "status": "REGISTERED",
            "creationDate": 1364293450.67,
            "description": "",
            "workflowType": {
                "version": "1",
                "name": "Crawl"
            }
        },
        {
            "status": "REGISTERED",
```

```
                "creationDate": 1364492094.968,
                "workflowType": {
                    "version": "1",
                    "name": "testW"
                }
            }
        ]
    }
```

create(*name*, *version*, *status='REGISTERED'*, *creation_date=0.0*, *deprecation_date=0.0*, *task_list=None*, *child_policy='TERMINATE'*, *execution_timeout='300'*, *decision_tasks_timeout='300'*, *description=None*, **args*, ***kwargs*)

Creates a new remote workflow type and returns the created WorkflowType model instance.

Parameters

- **name** (*String*) – name of the workflow type
- **version** (*String*) – workflow type version
- **status** (*swf.core.ConnectedSWFObject.{REGISTERED, DEPRECATED}*) – workflow type status
- **creation_date** (*float (timestamp)*) – creation date of the current WorkflowType
- **deprecation_date** (*float (timestamp)*) – deprecation date of WorkflowType
- **task_list** (*String*) – task list to use for scheduling decision tasks for executions of this workflow type
- **child_policy** (*CHILD_POLICIES.{TERMINATE | REQUEST_CANCEL | ABANDON}*) – policy to use for the child workflow executions when a workflow execution of this type is terminated
- **execution_timeout** (*String*) – maximum duration for executions of this workflow type
- **decision_tasks_timeout** (*String*) – maximum duration of decision tasks for this workflow type
- **description** (*String*) – Textual description of the workflow type

filter(*domain=None*, *registration_status='REGISTERED'*, *name=None*)

Filters workflows based on the `domain` they belong to, their `status`, and/or their `name`

Parameters

- **domain** (*swf.models.domain.Domain*) – domain the workflow type belongs to
- **registration_status** (*string*) – workflow type registration status to match, Valid values are: * `swf.constants.REGISTERED` * `swf.constants.DEPRECATED`
- **name** (*string*) – workflow type name to match

Returns list of matched WorkflowType models objects

Return type list

get(*name*, *version*)

Fetches the Workflow Type with *name* and *version*

Parameters

- **name** (*String*) – name of the workflow type
- **version** (*String*) – workflow type version

Returns matched workflow type instance

Return type swf.core.model.workflow.WorkflowType

A typical Amazon response looks like:

```
{
    "configuration": {
        "defaultExecutionStartToCloseTimeout": "300",
        "defaultTaskStartToCloseTimeout": "300",
        "defaultTaskList": {
            "name": "None"
        },
        "defaultChildPolicy": "TERMINATE"
    },
    "typeInfo": {
        "status": "REGISTERED",
        "creationDate": 1364492094.968,
        "workflowType": {
            "version": "1",
            "name": "testW"
        }
    }
}
```

get_or_create(*name*, *version*, *status='REGISTERED'*, *creation_date=0.0*, *deprecation_date=0.0*, *task_list=None*, *child_policy='TERMINATE'*, *execution_timeout='300'*, *decision_tasks_timeout='300'*, *description=None*)

Fetches, or creates the ActivityType with `name` and `version`

When fetching trying to fetch a matching workflow type, only name and version parameters are taken in account. Anyway, If you'd wanna make sure that in case the workflow type has to be created it is made with specific values, just provide it.

Parameters

- **name** (*String*) – name of the workflow type
- **version** (*String*) – workflow type version
- **status** (*swf.core.ConnectedSWFObject.{REGISTERED, DEPRECATED}*) – workflow type status
- **creation_date** (*float (timestamp)*) – creation date of the current WorkflowType
- **deprecation_date** (*float (timestamp)*) – deprecation date of WorkflowType
- **task_list** (*String*) – task list to use for scheduling decision tasks for executions of this workflow type
- **child_policy** (*CHILD_POLICIES.{TERMINATE | REQUEST_CANCEL | ABANDON}*) – policy to use for the child workflow executions when a workflow execution of this type is terminated
- **execution_timeout** (*String*) – maximum duration for executions of this workflow type
- **decision_tasks_timeout** (*String*) – maximum duration of decision tasks for this workflow type
- **description** (*String*) – Textual description of the workflow type

Returns Fetched or created WorkflowType model object

Return type WorkflowType

2.2.3 WorkflowExecutionQuerySet

class swf.querysets.workflow.**WorkflowExecutionQuerySet**(*domain*, **args*, ***kwargs*)
Fetches Workflow executions

all(*status='OPEN'*, *start_oldest_date=30*)
Fetch every workflow executions during the last *start_oldest_date* days, with *status*

Parameters

- **status** (*swf.models.WorkflowExecution.{STATUS_OPEN, STATUS_CLOSED}*) – Workflow executions status filter
- **start_oldest_date** (*integer (days)*) – Specifies the oldest start/close date to return.

Returns workflow executions objects list

Return type list

A typical amazon response looks like:

```
{
    "executionInfos": [
        {
            "cancelRequested": "boolean",
            "closeStatus": "string",
            "closeTimestamp": "number",
            "execution": {
                "runId": "string",
                "workflowId": "string"
            },
            "executionStatus": "string",
            "parent": {
                "runId": "string",
                "workflowId": "string"
            },
            "startTimestamp": "number",
            "tagList": [
                "string"
            ],
            "workflowType": {
                "name": "string",
                "version": "string"
            }
        }
    ],
    "nextPageToken": "string"
}
```

filter(*domain=None*, *status='OPEN'*, *tag=None*, *workflow_id=None*, *workflow_type_name=None*, *workflow_type_version=None*, **args*, ***kwargs*)
Filters workflow executions based on kwargs provided criteras

Parameters

- **domain_name** (*String*) – workflow executions attached to domain with provided domain_name will be kept
- **status** (*string*) – workflow executions with provided status will be kept. Valid values are: * `swf.models.WorkflowExecution.STATUS_OPEN` * `swf.models.WorkflowExecution.STATUS_CLOSED`

- **tag** (*String*) – workflow executions containing the tag will be kept
- **workflow_id** (*String*) – workflow executions attached to the id will be kept
- **workflow_type_name** (*String*) – workflow executions attached to the workflow type with provided name will be kept
- **workflow_type_version** (*String*) – workflow executions attached to the workflow type of the provided version will be kept

Be aware that querying over status allows the usage of statuses specific kwargs

•STATUS_OPEN

param start_latest_date latest start or close date and time to return (in days)

type start_latest_date int

•STATUS_CLOSED

param start_latest_date workflow executions that meet the start time criteria of the filter are kept (in days)

type start_latest_date int

param start_oldest_date workflow executions that meet the start time criteria of the filter are kept (in days)

type start_oldest_date int

param close_latest_date workflow executions that meet the close time criteria of the filter are kept (in days)

type close_latest_date int

param close_oldest_date workflow executions that meet the close time criteria of the filter are kept (in days)

type close_oldest_date int

param close_status must match the close status of an execution for it to meet the criteria of this filter. Valid values are: * `CLOSE_STATUS_COMPLETED` * `CLOSE_STATUS_FAILED` * `CLOSE_STATUS_CANCELED` * `CLOSE_STATUS_TERMINATED` * `CLOSE_STATUS_CONTINUED_AS_NEW` * `CLOSE_TIMED_OUT`

type close_status string

returns workflow executions objects list

rtype list

`get` (*workflow_id*, *run_id*)

2.2.4 ActivityTypeQuerySet

class `swf.querysets.activity.`**`ActivityTypeQuerySet`**(*domain*, **args*, ***kwargs*)

Swf activity type queryset object

Allows the user to interact with amazon's swf activity types through a django-queryset-like interface

Parameters **domain** (*swf.models.domain.Domain*) – domain the activity type belongs to

`all`(*registration_status='REGISTERED'*)

Retrieves every activity types

Parameters **registration_status** (*string*) – activity type registration status to match, Valid values are: * `swf.constants.REGISTERED` * `swf.constants.DEPRECATED`

Returns list of matched ActivityType models objects

Return type list

A typical Amazon response looks like:

```
{
    "nextPageToken": "string",
    "typeInfos": [
        {
            "activityType": {
                "name": "string",
                "version": "string"
            },
            "creationDate": "number",
            "deprecationDate": "number",
            "description": "string",
            "status": "string"
        }
    ]
}
```

create (*name*, *version*, *status='REGISTERED'*, *description=None*, *creation_date=0.0*, *deprecation_date=0.0*, *task_list=None*, *task_heartbeat_timeout=0*, *task_schedule_to_close_timeout=0*, *task_schedule_to_start_timeout=0*, *task_start_to_close_timeout=0*, **args*, ***kwargs*)

Creates a new remote activity type and returns the created ActivityType model instance.

Parameters

- **name** (*str*) – name of the ActivityType
- **version** (*str*) – version of the ActivityType
- **status** (*swf.constants.{REGISTERED, DEPRECATED}*) – ActivityType status
- **description** (*str | None*) – ActivityType description
- **creation_date** (*float (timestamp)*) – creation date of the current ActivityType
- **deprecation_date** (*float (timestamp)*) – deprecation date of ActivityType
- **task_list** (*str*) – specifies the default task list to use for scheduling tasks of this activity type.
- **task_heartbeat_timeout** (*int*) – default maximum time before which a worker processing a task of this type must report progress by calling RecordActivityTaskHeartbeat.
- **task_schedule_to_close_timeout** (*int*) – default maximum duration for a task of this activity type.
- **task_schedule_to_start_timeout** (*int*) – default maximum duration that a task of this activity type can wait before being assigned to a worker.
- **task_start_to_close_timeout** (*int*) – default maximum duration that a worker can take to process tasks of this activity type.

filter (*domain=None*, *registration_status='REGISTERED'*, *name=None*)

Filters activity types based on their status, and/or name

Parameters

- **domain** (*swf.models.domain.Domain*) – domain the activity type belongs to
- **registration_status** (*string*) – activity type registration status to match, Valid values are: * `swf.constants.REGISTERED` * `swf.constants.DEPRECATED`
- **name** (*string*) – activity type name to match

Returns list of matched ActivityType models objects

Return type list

get (*name*, *version*)

Fetches the activity type with provided `name` and `version`

Parameters

- **name** (*String*) – activity type name to fetch
- **version** (*String*) – activity version to fetch

Returns Matched activity type instance

Return type swf.models.activity.ActivityType

A typical Amazon response looks like:

```
{
    "configuration": {
        "defaultTaskHeartbeatTimeout": "string",
        "defaultTaskList": {
            "name": "string"
        },
        "defaultTaskScheduleToCloseTimeout": "string",
        "defaultTaskScheduleToStartTimeout": "string",
        "defaultTaskStartToCloseTimeout": "string"
    },
    "typeInfo": {
        "activityType": {
            "name": "string",
            "version": "string"
        },
        "creationDate": "number",
        "deprecationDate": "number",
        "description": "string",
        "status": "string"
    }
}
```

get_or_create (*name*, *version*, *status='REGISTERED'*, *description=None*, *creation_date=0.0*, *deprecation_date=0.0*, *task_list=None*, *task_heartbeat_timeout=0*, *task_schedule_to_close_timeout=0*, *task_schedule_to_start_timeout=0*, *task_start_to_close_timeout=0*)

Fetches, or creates the ActivityType with `name` and `version`

When fetching trying to fetch a matching activity type, only name and version parameters are taken in account. Anyway, If you'd wanna make sure that in case the activity type has to be created it is made with specific values, just provide it.

Parameters

- **name** (*str*) – name of the ActivityType
- **version** (*str*) – version of the ActivityType

- **status** (*swf.constants.{REGISTERED, DEPRECATED}*) – ActivityType status
- **description** (*str | None*) – ActivityType description
- **creation_date** (*float (timestamp)*) – creation date of the current ActivityType
- **deprecation_date** (*float (timestamp)*) – deprecation date of ActivityType
- **task_list** (*str*) – specifies the default task list to use for scheduling tasks of this activity type.
- **task_heartbeat_timeout** (*int*) – default maximum time before which a worker processing a task of this type must report progress by calling RecordActivityTaskHeartbeat.
- **task_schedule_to_close_timeout** (*int*) – default maximum duration for a task of this activity type.
- **task_schedule_to_start_timeout** (*int*) – default maximum duration that a task of this activity type can wait before being assigned to a worker.
- **task_start_to_close_timeout** (*int*) – default maximum duration that a worker can take to process tasks of this activity type.

Returns Fetched or created ActivityType model object

Return type ActivityType

2.3 Actors

2.3.1 Actor

class swf.actors.core.**Actor**(*domain*, *task_list*)

SWF Actor base class

Actor is running through a thread in order for it's polling operations not to be blocking. Many actors might be ran through the same process.

Usage example: implementing an activity worker or a decider using an actor is the typical usage

Parameters

- **domain** (*swf.models.Domain*) – Domain the Actor should interact with
- **task_list** (*string*) – task list the Actor should watch for tasks on

start()

Launches the actor

Any class overriding actor's class should set this method to update actor's status to Actor.STATES.RUNNING

stop()

Stops the actor

Sets actor's status to Actor.STATES.STOPPED, and waits for the last polling operation to end before shutting down.

2.3.2 ActivityWorker

class swf.actors.worker.**ActivityWorker**(*domain*, *task_list*)
Activity task worker actor implementation

Once started, will start polling for activity task, to process, and emitting heartbeat until it's stopped or crashes for some reason.

Parameters

- **domain** (*swf.models.Domain*) – Domain the Actor should interact with
- **task_list** (*string*) – task list the Actor should watch for tasks on
- **last_token** (*string*) – last seen task token

cancel(*task_token*, *details=None*)
Responds to swf that the activity task was canceled

Parameters

- **task_token** (*string*) – canceled activity task token
- **details** (*string*) – provided details about cancel

complete(*task_token*, *result=None*)
Responds to ‘‘swf‘ that the activity task is completed

Parameters

- **task_token** (*string*) – completed activity task token
- **result** (*string*) – The result of the activity task.

fail(*task_token*, *details=None*, *reason=None*)
Replies to swf that the activity task failed

Parameters

- **task_token** (*string*) – canceled activity task token
- **details** (*string*) – provided details about cancel
- **reason** (*string*) – Description of the error that may assist in diagnostics

heartbeat(*task_token*, *details=None*)
Records activity task heartbeat

Parameters

- **task_token** (*string*) – canceled activity task token
- **details** (*string*) – provided details about cancel

poll(*task_list=None*, ***kwargs*)
Polls for an activity task to process from current actor's instance defined task_list

if no activity task was polled, raises a PollTimeout exception.

Parameters **task_list** (*string*) – task list the Actor should watch for tasks on

Raises PollTimeout

Returns polled activity task

Type swf.models.ActivityTask

start()
Launches the actor

Any class overriding actor's class should set this method to update actor's status to Actor.STATES.RUNNING

stop()
Stops the actor

Sets actor's status to Actor.STATES.STOPPED, and waits for the last polling operation to end before shutting down.

PYTHON MODULE INDEX