kissats Documentation

Release 1.0.0a6

Bob Folkes

Jul 29, 2018

Contents

A flexible Automated Test System to simplify testing and resource management.

Contents:

CHAPTER 1

Readme

1.1 Known issues

- They are still hiding from me

1.2 Future Improvements

- Python 3 support
- Multi-threading support
- Multi-processing support

CHAPTER 2

Schemas

Schemas are based on the format defined by Cerberus

2.1 Global Parameters

```
# importable test/task package name, consumed on use
task_package:
    type: string
    required: True
    default: null
    nullable: True
# PIP compatible version string IE: >=1.0.0, consumed on use
package_version:
    type: string
    default: null
    nullable: True
# test groups to execute, consumed on use
test_groups:
    type: list
    schema:
        type: string
    default: []
# ATS client for communication with the ATS resource manager, consumed on use
ats_client:
    type: string
    default: null
    nullable: True
# any string not listed here will be considered a failure
valid_result:
    type: list
    schema:
        type: string
    default: [Passed, Completed, Skipped]
```

(continues on next page)

(continued from previous page)

```
    required: True
# Max number of threads to use in threading mode, consumed on use
thread_limit:
    type: integer
    default: 5
    required: True
# Max number of processes to use in sub_process mode, consumed on use
process_limit:
    type: integer
    default: 5
    required: True
# Resource reservation mode, consumed on use
resource_mode:
    type: string
    allowed: [all, per_task_separate, per_task_combine, custom_all, custom_separate,␣
↪custom_combine]
    default: all
    required: True
# The Automated Test System
ats:
    type: string
    required: True
    default: null
    nullable: True
# The Device Under Test
dut:
    type: string
    required: True
    default: null
    nullable: True
# When ignore_prereq is set, all prereq checking will be ignored
ignore_prereq:
    type: boolean
    required: True
    default: False
# kwargs to be passed to an auto ATS client
ats_client_kwargs:
    type: dict
    required: True
    default: null
    nullable: True
```

2.2 Task Parameters

```
# task name, typicly the module __name__
name:
    type: string
    required: True
# short description of the task
description:
    type: string
    required: True
# if this task fails, stop all further testsing
stop_suite_on_fail:
```

(continues on next page)

(continued from previous page)

```
    type: boolean
    default: True
    required: True
#
exclusive_resources:
    type: list
    default: []
    schema:
        type: string
    required: True
#
shared_resources:
    type: list
    default: []
    schema:
        type: string
    required: True
# configurations to be passed to the resource manager
# for each resource. Key name must match a resource,
# the value will be passed to the resource manager
resource_configs:
    type: dict
    default: {}
    required: True
# if resources are busy, max number of times to retry
max_resource_retry:
    type: integer
    default: 5
    required: True
#
thread_safe:
    type: boolean
    default: False
    required: True
#
process_safe:
    type: boolean
    default: False
    required: True
#
valid_ats:
    type: list
    default: [any]
    schema:
        type: string
    required: True
#
valid_duts:
    type: list
    default: [any]
    schema:
        type: string
    required: True
# keys required to be present in the global
# parameter dictionary
req_param_keys:
    type: list
```

(continues on next page)

(continued from previous page)

```
    default: []
    schema:
        type: string
    required: True
# optional keys that will be used if present
# in the global parameter dictionary
optional_param_keys:
    type: list
    default: []
    schema:
        type: string
    required: True
# tasks that must successfully run before this task can run
prereq_tasks:
    type: list
    default: []
    schema:
        type: string
    required: True
# All prereq_tasks must have been run in the same thread/process
prereq_same_thread:
    type: boolean
    default: True
    required: True
# estimated test time in seconds (including optional
# task_setup and task_teardown execution time)
est_task_time:
    type: integer
    default: 3600
    required: True
# if true, will always run teardown regardless of completion
# status of setup or run
always_teardown:
    type: boolean
    default: False
    required: True
# other data to pass to the class, must be JSON
# serializable if thread_safe or process_safe are true
extra_metadata:
    type: dict
    default: null
    nullable: True
# priorty level of the task, lower is given more
# prefence in multiprocess and threading modes
priority:
    type: integer
    default: 5
    required: True
```

2.3 Task Return

```
# result of the task, see global param
# schema for default non-failure strings
result:
```

(continues on next page)

(continued from previous page)

```
    type: string
    required: True
# optional dictionary of extra metadata to be reported
metadata:
    type: dict
    required: True
    default: {}
    schema:
        message:
            type: string
            required: True
            empty: True
            default: ''
# optional key for reporting multiple test results
# via the registered reporting function
multi_result:
    type: list
    schema:
        type: dict
        schema:
            name:
                type: string
                required: True
            description:
                type: string
                required: True
                default: ''
            result:
                type: string
                required: True
            metadata:
                type: dict
                required: True
                default: {}
                schema:
                    message:
                        type: string
                        required: True
                        empty: True
                        default: ''
```

2.4 Reporting

```
name:
    type: string
    required: True
description:
    type: string
    required: True
result:
    type: string
    required: True
metadata:
    type: dict
```

(continues on next page)

(continued from previous page)

```
    schema:
        run_time:
            type: float
            default: 0
            required: True
        est_task_time:
            type: float
            default: 0
            required: True
        run_time_delta:
            type: float
            default: 0
            required: True
    required: True
```

CHAPTER 3

Task Pack

A Task Pack is a class that manages a group of setup tasks, teardown tasks, tests and resources.

3.1 Property definitions

3.1.1 Resource Run Modes

Note:

- The resource mode only applies if an ATS Client has been registered.
- For all custom modes, a scheduler function must be regesiterd with schedule_func before calling any run que methods.

Warning: "all" is the only mode currently implemented

- **all** Will reserve all resources needed by all tasks for the estimated duration of the run. **This is the default mode**

  Note: run_all_que method must be used for running tasks

- **per_task_separate** Will reserve resources on a task by task basis as each task is run. Each task will hold its own set of resources.
- **per_task_combine** Will reserve resources on a task by task basis as each task is run. Resources common to tasks will be consolidated so they use the same resource.
- **custom_all** The test que will be passed to the function registered with schedule_func for ordering. Resource management will be handled the same as **all**

- **custom_separate** The que selected for execution will be passed to the function registered with schedule_func for ordering. Resource management will be handled the same as **per_task_separate**
- **custom_combine** The que selected for execution will be passed to the function registered with schedule_func for ordering. Resource management will be handled the same as **per_task_combine**

3.1.2 ats_client

ats_client module containing the ATS_Client class to import and instantiate for resource management, if None, no client will be used.

Note: if set to "auto", we will attempt to first import an ATS client based on the ATS name provided in the init_params, if that fails, we will attempt to pip install, then import, if that fails a KissATSError will be raised

---

3.2 TaskPack Class

class kissats.task_pack.**TaskPack**(*init_params=None*, *schema_add=None*, *que_class=<class 'kissats.queues.PackQues'>*)

Bases: *kissats.task_pack.PackParams*

Schedules and executes a group or package of kissats.BaseTask

see *Global Parameters*

Parameters

- **init_params** (*dict*) – Initialization parameter dict. see XXXXX
- **schema_add** (*dict*) – (Optional) Additional Cerberus schema definition to be applied to the init_params
- **que_class** (*object*) – Class or object containing the queues, see `BaseQues`

Note: If task_package is not supplied in the init_params, tasks must be added using the appropriate add task method. Any method that depends on a valid task_package will raise.

---

add_setup_task(*task*, *allow_dupe=False*, *top=False*)

add a task to the setup queue

Parameters

- **task** (*kissats.task.Task or str or ModuleType*) – Task to add
- **allow_dupe** (*bool*) – Allow the task to run multiple times
- **top** (*bool*) – Place the task at the top of the queue

see *add_test_task()* for Task input handling and further allow_dupe explanation

Returns True if in the queue

Return type bool

add_teardown_task(*task*, *allow_dupe=False*, *top=False*)

add a task to the teardown queue

Parameters

- **task** (*kissats.task.Task or str or ModuleType*) – Task to add
- **allow_dupe** (*bool*) – Allow the task to run multiple times
- **top** (*bool*) – Place the task at the top of the queue

see *add_test_task()* for Task input handling and further allow_dupe explanation

Returns True if in the queue

Return type bool

Raises

- FailedPrereq
- KissATSError

add_test_group(*test_group*)
Add all tests in the test group to the test queue

If a corresponding group specific setup and teardown exists, they will also be added to the appropriate queue.

Parameters **test_group** (*str*) – Test group to add to the test queue.

Note: There must be a corresponding get_<test_group>_tests function in the test package's seq_test

---

Raises TaskPackageNotRegistered

add_test_task(*task*, *allow_dupe=False*, *top=False*)
add a task to the test queue

Parameters

- **task** (*kissats.task.Task or str or ModuleType*) – Task to add
- **allow_dupe** (*bool*) – Allow the task to run multiple times If set to false and the task is already in the queue a warning will be logged and processing will continue.
- **top** (*bool*) – Place the task at the top of the queue

Warning: If dut and/or ats are not set in the TaskPack, the dut and/or ats will not be verified and the task will be added to the queue

---

Note: Task input handling:

- If task is a kissats.task.Task based class it will be added directly to the queue
- If task is a str we will an attempt will be made to import by the Task class, if the import fails, we will prepend with the package name and try again.
- If task is a ModuleType, it will be passed directly to the Task class
- If the dut from the global params is not listed in the task params key valid_duts the task will not be added and processing will continue
- If the ats from the global params is not listed in the task params key valid_ats the task will not be added and processing will continue

Returns True if in the queue

Return type bool

all_resources
A list of all resources needed for tasks currently in any queue

Resource should be kissats.ResourceReservation

ats
The Automated Test System used to perform the testing

ats_client
The ATS client for communication with the ATS

call_scheduler()
Call the registered scheduler/optimizer

Passes all queues to a previously registered function

check_prereqs(*task*)
Check if prereq tasks have been completed for a task

Parameters **task** (Task) – the task to check

Returns

- (list): prereqs needed
- (list): failed prereqs

Return type tuple

clear_all_que()
clear all task queues

clear_delay_que()
clear the delay queue

clear_setup_que()
clear the setup queue

clear_teardown_que()
clear the teardown queue

clear_test_que()
clear the test queue

completed_tasks
dict of completed tasks and their results

dut
The Device Under Test

est_run_time
Estimated total run time in seconds

get_seq_group(*group_name*, *seq_name*)
Get a list of tasks from a seq

Parameters

- **group_name** (*str*) – Group to find

- **seq_name** (*str*) – seq to check (setup, test or teardown)

Returns List of tasks

Return type list

ignore_prereq
when set, will ignore prereqs

json_params
Keys in the parameter dictionary formated in JSON

Note: Any key that cannot be flattened by json.dumps will be excluded

params
The global parameter dict

process_limit
Max additional processes to use

report_func
Function to report results

report_result(*result*)
Report results using a registered reporting function.

If no reporting function is registered, result will be reported using the python built in logging module.

Parameters **result** (*dict*) – see reporting_schema for details

resource_mode
Resource reservation mode

run_all_que()
Run all queue's

If ATS client is registered and resource mode is set to "all" or "custom_all", all resources will be reserved and claimed. When complete, all resources will be released.

If resource mode is set to a custom mode, the registered schedular function will be called before execution.

Will run all queue's in order:

- setup
- test
- teardown

Raises

- KissATSError
- ResourceUnavailable

run_mode
The global run mode, normal, process or thread

run_setup_que()
run all tasks in setup queue

run_teardown_que()
run all tasks in the teardown queue

run_test_que()
run all tasks in test queue

schedule_func
Function to schedule/order tasks

setup_list
The list of setup tasks required by the task package.

Will call the get_global_setup function from the seq_setup module in the task_package to populate the list.

task_pack
The Python package containing the tasks to run

Also accepts a wheel, distribution name must match the import name!

Note: Don't be like PyYAML: distribution name is PyYAML import name is yaml

---

Warning: This property can only be set once!

teardown_list
The list of teardown tasks required by the task package

Will call the get_global_teardown function from the seq_teardown module in the task_package to populate the list.

test_groups
The scheduled test groups

thread_limit
Max additional threads to use

valid_result
Valid result returns

These are the only result values that will be considered a non-failure condition.

3.3 PackParams Class

class kissats.task_pack.**PackParams**(*init_params=None*, *schema_add=None*)
Bases: object

Holds the parameters of the task package

see *Global Parameters*

Parameters

- **init_params** (*dict*) – Initialization parameter dict. see XXXXX
- **schema_add** (*dict*) – (Optional) Additional Cerberus schema definition to be applied to the init_params

---

Note: If task_package is not supplied in the init_params, tasks must be added using the appropriate add task method. Any method that depends on a valid task_package will raise.

---

all_resources
A list of all resources needed for tasks currently in any queue

Resource should be kissats.ResourceReservation

ats
The Automated Test System used to perform the testing

ats_client
The ATS client for communication with the ATS

completed_tasks
dict of completed tasks and their results

dut
The Device Under Test

est_run_time
Estimated total run time in seconds

ignore_prereq
when set, will ignore prereqs

json_params
Keys in the parameter dictionary formated in JSON

Note: Any key that cannot be flattened by json.dumps will be excluded

---

params
The global parameter dict

process_limit
Max additional processes to use

report_func
Function to report results

resource_mode
Resource reservation mode

run_mode
The global run mode, normal, process or thread

schedule_func
Function to schedule/order tasks

setup_list
The list of setup tasks required by the task package.

Will call the get_global_setup function from the seq_setup module in the task_package to populate the list.

task_pack
The Python package containing the tasks to run

Also accepts a wheel, distribution name must match the import name!

Note: Don't be like PyYAML: distribution name is PyYAML import name is yaml

---

> **Warning:** This property can only be set once!

teardown_list
The list of teardown tasks required by the task package

Will call the get_global_teardown function from the seq_teardown module in the task_package to populate the list.

test_groups
The scheduled test groups

thread_limit
Max additional threads to use

valid_result
Valid result returns

These are the only result values that will be considered a non-failure condition.

CHAPTER 4

Task

Kiss ATS Task

class kissats.task.**BaseTask**(*task_in=None*, *global_params_in=None*, *schema_add=None*)

Bases: object

Base task class

Parameters

- **task_in** (*Any*) – an object containing the appropriate functions
- **global_params_in** (*dict*) – Global dictionary of parameters used to configure the environment. This dictionary will also be passed to the registered task functions.
- **schema_add** (*dict*) – additional schema definition to be applied to task params

ats_client

ATS client class based on BaseATSClient

check_ats_valid()

check if task is valid for the ATS

check_dut_valid()

check if task is valid for the DUT

check_requires()

Verify all requirements for executing the task are met

Returns True if all requirements are met

Return type bool

check_resources_ready()

check if all resources are reserved for the task

Returns True if all resources are ready

Return type bool

claim_resources()
Claim all reservations

global_params
Global Parameters to be passed to the task

missing_keys
Missing parameter dictionary keys

name
The name of the task. This will be the name tracked and reported by the TaskPack

params
Run parameters of the task.

see *Task Parameters*.

release_resources()
Release all reservations and clear time window

reserve_resources()
Request reservations for all resources

Returns True if all resources were reserved

Return type bool

resource_delay()
delay all resource reservations

Returns True if all resources are reserved

Return type bool

resource_list
List of resources needed for task

run_task()
run the task

If the task module has a task_setup, task_setup will be executed first.

If the task module has a task_teardown, task_teardown will be executed after the task_main function. If the task params key always_teardown is set to True, task_teardown will run regardless of the exit status of task_setup and the run function.

Warning: If the property always_teardown is True, task_teardown will execute even if task_setup or run raise an exception.

set_time_window(*start_time*, *end_time*)
set the expected execution time of the task for reservation planning

Parameters

- **start_time** (*float*) – Epoch time of expected start Default: time.time() of function call
- **end_time** (*float*) – Epoch time of expected completion. Default: start_time + time_estimate

task_main()
The main task

task_prereqs
prereqs for the task

task_setup()
Setup action for this task.

task_teardown()
Teardown action for this task

time_estimate
estimated total run time

time_window
Planned execution time

class kissats.task.**Task**(*task_in=None*, *global_params_in=None*, *schema_add=None*)
Bases: *kissats.task.BaseTask*

a task and it's parameters

Parameters

- **task_in** (*Any*) – The importable name of the task to run or a module containing the appropriate functions

  Note: If a str, should be in package.module format

- **global_params_in** (*dict*) – Global dictionary of parameters used to configure the environment. This dictionary will also be passed to all registered task functions.
- **schema_add** (*dict*) – (Optional) Additional Cerberus schema definition to be applied to the init_params

ats_client
Instantiated ATS client class based on BaseATSClient

check_ats_valid()
check if task is valid for the ATS

If ATS is not specified in global params this method will return True

check_dut_valid()
check if task is valid for the DUT

If DUT is not specified in global params this method will return True

check_requires()
Verify all requirements for executing the task are met

Returns True if all requirements are met

Return type bool

Raises

- MissingTestParamKey
- InvalidDut
- InvalidATS
- ResourceNotReady

check_resources_ready()
check if all resources are reserved for the task

Returns

True if an ATS Client is registered and all resources are reserved, will also return True if an ATS Client is not registered

Return type bool

claim_resources()
Claim all reservations

get_params()
The parameters for executing the task

global_params
Global Parameters to be passed to the task

missing_keys
Missing parameter dictionary keys

name
The name of the task.

This will be the name tracked and reported by the TaskPack

params
Run parameters of the task.

This will call get_params on the task module registered, expecting a dict conforming with the *Task Parameters*.

release_resources()
Release all reservations and clear time window

reserve_resources()
Request reservations for all resources

Returns True if all resources were reserved

Return type bool

resource_delay()
Delay all resource reservations

> **Warning:** this method will reset the time_window

Returns True if all resources are reserved

Return type bool

resource_list
List of resources needed for task

run_task()
run the task

If the task module has a task_setup, task_setup will be executed first.

If the task module has a task_teardown, task_teardown will be executed after the run function. If the task params key always_teardown is set to True, task_teardown will run regardless of the exit status of task_setup and the run function.

> **Warning:** If the class property always_teardown is True, task_teardown will execute even if task_setup or task_main raise an exception.

set_time_window(*start_time=None*, *end_time=None*)
set the expected execution time of the task for reservation planning

Parameters

- **start_time** (*float*) – Epoch time of expected start Default: time.time() of function call
- **end_time** (*float*) – Epoch time of expected completion. Default: start_time + time_estimate

task_main()
The main task function

task_mod
Any class, module or duck

Must contain the minimum task execution attributes run and get_params.

Alternately accepts a string and will import the module.

Note: If a string is used, it should be in importable package.module format.

---

task_prereqs
prereqs for the task

task_setup()
Setup action for this task.

task_teardown()
Teardown action for this task

time_estimate
estimated total run time

time_window
Planned execution time

CHAPTER 5

Sample Task

see *Task Parameters* and *Task Return* in *Schemas*

```
"""A sample task for Kiss ATS"""
# TODO(BF): Needs updating

import logging


logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())


def get_params(global_params):
    """The parameters for executing the task

    An example implementation:
        If the task requires a 32-bit Linux PC running centOS and the ATS
        manager has the capability to configure resources.
        Some of the params might look like:
             params['exclusive_resources'] = ['linux_pc']
             params['resource_config'] = {'linux_pc':['32-bit', 'centOS']}

        The object contained in the key "linux_pc" will be flattened and sent
        to the ATS manager via the defined ATS client.  The ATS manager would
        schedule the configuration of a resource with a 32 bit installation of
        centOS.  The ATS manager would return a pre-reservation ID and a time
        when the resource will be ready.  The task_pack will delay the test/task
        until the resource is configured/ready.  Depending on the resource_mode
        selected task_pack will continue with other test/task actions while
        waiting or wait for the resource to be ready.

    required params keys:
        * name
        * description
```

(continues on next page)

(continued from previous page)

```
    """

    params = dict()
    # required keys
    params['name'] = __file__
    params['description'] = __doc__
    # optional keys, values listed are defaults
    params['stop_suite_on_fail'] = False
    params['exclusive_resources'] = list()
    params['shared_resources'] = list()
    params['max_resource_wait'] = int()
    params['max_resource_retry'] = int()
    params['thread_safe'] = False
    params['process_safe'] = False
    params['valid_ats'] = list()
    params['valid_duts'] = list()
    params['req_param_keys'] = list()
    params['optional_param_keys'] = list()
    params['prereq_tasks'] = list()
    params['est_test_time'] = int()
    params['always_teardown'] = False
    params['priority'] = 5
    params['extra_metadata'] = None

    return params


def task_setup(global_params):
    """Setup action for this task.

    required return value is None

    If the function encounters a condition that needs to stop all
    testing or task execution an exception must be raised.

    Warning:
        Setup actions are NOT tests and should not test.
        Setup conditions should be verified before returning
        from the function and if not met an exception should
        be raised to halt testing.

    """

    return


def task_main(global_params):
    """The main task function

    If the valid_ats is a valid kiss ats available on pypi or
    already installed, the global_params will contain an instantiated client
    with all needed resources claimed.

    required return is a dict containing at least
    the "result" and "metadata" keys
```

(continues on next page)

(continued from previous page)

```
    An optional additional key 'multi_result' is permitted.
    multi_result must be a list of dictionaries containing the "name",
    "description", "result" and "metadata" keys.
    The items in the list will be reported in the order they
    are contained in the list.

    """

    result = "Passed"
    task_message = ""

    # Multi result is an optional return dictionary item
    multi_result = list()

    multi_result.append({'name': "sub_task", 'description': "sub_description",
                         'result': "sub_result", 'metadata': "sub_task_message"})

    return {'result': result, 'metadata': task_message, 'multi_result': multi_result}


def task_teardown(global_params):
    """Teardown action for this task.

    required return value is None

    If the function encounters a condition that needs to stop all
    testing or task execution an exception must be raised.

    Warning:
        Teardown actions are NOT tests and should not test.
        Post test conditions should be verified before returning
        from the function and if not met an exception should
        be raised to halt testing.

    """

    return
```

CHAPTER 6

ATS Client

The base ATS Client class(es) for KISS ATS

class kissats.ats_client.**BaseATSClient**

Bases: object

Base ATS Client Class

ats_server

The address of the ATS Server

claim_reservation(*pre_reservation_id*)

Claim a pre-reserved resource.

- If the reservation is claimed late and will not be available for the entire reservation_duration an exception will be raised.
- If claimed late and another slot is available a new pre-reservation will be provided, see Returns below

Parameters **pre_reservation_id** (*str*) – The pre_reservation_id returned by get_future_reservation

Returns

- **Min keys if success**
  - reservation_id(str): if the reservation is a success.
  - expire_time(float): epoch time when the reservation expires
  - resource_config(object): Current configuration of the resource
- **Min keys if failure**
  - pre_reservation_id (str): new pre_reservation ID
  - new_avail (float): New available time
  - new_expire (float): New expiration time

Return type dict

Raises `ResourceUnavailable`

get_all_resources()
Get a list of all resources managed by the ATS

Returns List of all resources managed by the ATS

Return type list

get_available_resources()
Get a list of available resources

Returns List of available resources

Return type list

get_resource_availablity(*resource*, *start_time=None*, *end_time=None*)
Get the time when a resource will become available.

If the resource is not available at the time requested, avail_start and avail_end will be the soonest time slot available.

Parameters

- **resource** (*str*) – name of resource
- **start_time** (*float*) – Epoch time in seconds
- **end_time** (*float*) – Epoch time in seconds

Returns

min keys:

- available (bool) True if available at the time requested
- avail_start (float)
- avail_end (float)

Return type dict

get_resource_config(*resource*)
Get the current configuration of a resource

Parameters **resource** (*str*) – name of resource

Returns the current configuration of the resource

Return type object

release_resource(*reservation_id*)
Release a previously reserved resource

Parameters **reservation_id** (*str*) – reservation_id or pre-reservation_id of resource

Returns True if released

Return type bool

request_reservation(*resource*, *res_config=None*, *time_needed=None*, *reservation_duration=3600.0*, *next_available=True*, *reservation_mode='exclusive'*)
Request resource reservation with an optional configuration.

this will put a preliminary lock on the resource, the final lock must be requested after the time_available using claim_reservation

Parameters

- **resource** (*str*) – The name of the resource requested
- **res_config** (*object*) – An object that can be serialized for transmission to the server. This optional object will define the requested configuration.
- **time_needed** (*float*) – time.time time the resource is needed. if not provided, default is now
- **reservation_duration** (*float*) – seconds the resource is requested for defaults to 3600 (1 hour)
- **next_available** (*bool*) – If the requested time_needed is not available, request the next available time.
- **reservation_mode** (*str*) – "exclusive" or "shared", default "exclusive"

Returns

- (str): UUID of pre-reservation.
- (float): epoch time resource will be available with the requested configuration.
- (float): epoch time the pre_reservation_ID will expire.

Return type tuple

Raises `ResourceUnavailable`

server_communicate(*server_request*)

Send a command to the server and return the server reply

Parameters **server_request** (*dict*) – the request to be sent with a key "command", all other keys will be placed in the extra data

Returns Dictionary of unflattened reply

Return type dict

Raises `ServerCommandMissing`

CHAPTER 7

ATS Resource

Base resource class

class kissats.ats_resource.**ResourceReservation**(*resource_name*, *ats_client*, *mode='exclusive'*, *max_retry=5*, *max_wait=None*)

Bases: object

An ATS Resource ...

Parameters

- **resource_name** (*str*) – The name of the resource
- **ats_client** (BaseATSClient) – ATS client class for communication to the ATS reservation system
- **mode** (*str*) – "exclusive" or "shared", default "exclusive"
- **max_retry** (*int*) – Max number of time to attempt to reserve the resource before raising an exception
- **max_wait** (*float*) – Max time to wait for the resource to become available before raising an exception

add_retry_count()

Add another retry to the counter

if retry count exceeds max retry, will raise

Raises ResourceRetryExceeded

claim_reservation()

Claim reservation

Returns True if successful

Return type bool

end_time

Epoch end time of the reservation

first_request_time
Epoch time of the first request to reserve the resource

get_next_avail_time()
Get the epoch time when a resource will become available.

> **Warning:** Does not reserve or claim the resource

Returns

min keys:

- available (bool) True if available at the time requested
- avail_start (float)
- avail_end (float)

Return type dict

max_retry
Max number of times to attempt to reserve the resource

max_wait_time
Max amount of time in seconds to wait for the resource to become available.

> **Warning:** If set to None, will wait indefinitely

name
Name of the resource

pre_res_expire
Epoch expiration time of the pre_reservation_id

pre_reservation_id
ID returned by a successful reservation

- If reservation is claimed, value is None
- If no reservation has been requested, value is None

release_reservation()
Release the current reservation or claim

request_reservation(*new_start_time=None*, *new_end_time=None*, *next_available=True*)
request a reservation

reservation_id
ID of the currently claimed reservation.

If not currently reserved and claimed, value is None

reservation_mode
Reservation mode. IE – exclusive or shared

resource_config
The current or requested configuration of the resource

- If the resource has not been reserved, the configuration to request.
- If the resource is reserved the actual configuration returned by the ATS

retry_count
: Number of unsuccessful attempts to reserve or claim the resource

start_time
: Epoch start time of the reservation

CHAPTER 8

Sample Resource

Todo: Create some sample resources

```
"""Some sample resources"""


import logging


logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())
```

CHAPTER 9

Common

Some common helper functions

kissats.common.**load_data_file**(*file_location*)
Load a schema from a .json or .yaml file

Parameters **file_location** (*string or pathlib.Path*) – Absolute location of the data file to load.

Returns data file contents

Return type dict

kissats.common.**pip_in_package**(*package_to_pip*)
Pip install a package into the current environment

CHAPTER 10

Queues

Package queue handlers

class kissats.queues.**BaseQues**
Bases: object

Base class for task management queues

active_que_len
length of the active queue

add_to_active_que(*task*, *top=False*)
add a task to the active queue

clear_active_que()
Clear the active queue

clear_all_que()
clear all queues

clear_delay_que()
Clear the delay queue

delay_que_add_task(*task*)
place task(s) and reservation(s) in delay queue ordered on reservation time

Parameters **task** (Task) – Task to add

delay_que_len
length of the delay queue

in_active_que(*task*)
Check if a task is already in the active queue

Parameters **task** (Task) – The task to check

peek_delay()
peek at the right side (bottom)

pop_active()
pop the next item from the right side (bottom)

popleft_active()
pop the next item from the left side (top)

remove_from_active_que(*task*)
remove a task from the active queue

Parameters **task** (*object*) – task to remove

set_active_que(*que_name*)
Set the active queue

Parameters **que_name** (*str*) – queue to set active

class kissats.queues.**PackQues**
Bases: *kissats.queues.BaseQues*

Task manager queues

active_que_len
length of the active queue

add_to_active_que(*task*, *top=False*)
add a task to the active queue

clear_active_que()
Clear the active queue

clear_all_que()
clear all queues

clear_delay_que()
Clear the delay queue

delay_que_add_task(*task*)
place task(s) and reservation(s) in delay queue ordered on reservation time

Parameters **task** (Task) – Task to add

delay_que_len
length of the delay queue

in_active_que(*task*)
Check if a task is already in the active queue

Parameters **task** (Task) – The task to check

Returns True if present

Return type bool

peek_delay()
peek at the right side (bottom)

pop_active()
pop the next item from the right side (bottom)

pop_delay()
pop the next item from the right side (bottom)

popleft_active()
pop the next item from the left side (top)

remove_from_active_que(*task*)
remove a task from the active queue

Parameters **task** (*object*) – task to remove

set_active_que(*que_name*)
Set the active queue

Parameters **que_name** (*str*) – Queue to set active

Raises KissATSError – If queue name is invalid

CHAPTER 11

Schema Handler

Schema definitions

class kissats.schemas.schemas.**MasterSchemaDirectory**
Bases: object

Master Schema directory

base_schema_location
location of the schema dir

global_param_schema
Global param schema

reporting_schema
The reporting schema

task_param_schema
Task param schema

task_return_schema
Task return schema

kissats.schemas.schemas.**load_schema**(*schema_location*)
Load a schema from a .json or .yaml file

Parameters **schema_location** (*string or pathlib.Path*) – Absolute location of the schema file to load.

Returns The schema in dict format

Return type dict

kissats.schemas.schemas.**normalize_and_validate**(*dict_to_check*, *schema*)
Normalize and validate a dictionary, will raise if invalid

Parameters

- **dict_to_check** (*dict*) – dictionary to check
- **schema** (*dict*) – schema to use

Returns Normalized and valid dictionary

Return type dict

Raises `SchemaMisMatch`

CHAPTER 12

Exceptions

Kiss ATS Exceptions

exception kissats.exceptions.**CriticalTaskFail**
Bases: *kissats.exceptions.KissATSError*

A critical task has failed.

exception kissats.exceptions.**FailedPrereq**
Bases: *kissats.exceptions.KissATSError*

Task has a failed prereq task.

exception kissats.exceptions.**InvalidATS**
Bases: *kissats.exceptions.KissATSError*

Invalid ATS selected for task.

exception kissats.exceptions.**InvalidConfigRequest**
Bases: *kissats.exceptions.KissATSError*

An invalid request to reconfigure a resource was made.

exception kissats.exceptions.**InvalidDataFile**
Bases: *kissats.exceptions.KissATSError*

An invalid data file was requested

exception kissats.exceptions.**InvalidDut**
Bases: *kissats.exceptions.KissATSError*

Invalid DUT selected for task.

exception kissats.exceptions.**InvalidResourceMode**
Bases: *kissats.exceptions.KissATSError*

Invalid resource mode selected.

exception kissats.exceptions.**InvalidSchemaFile**
Bases: *kissats.exceptions.KissATSError*

An invalid schema file was requested

exception kissats.exceptions.**InvalidTask**
Bases: *kissats.exceptions.KissATSError*

Invalid task requested.

exception kissats.exceptions.**KissATSError**
Bases: exceptions.Exception

Base exception for package.

exception kissats.exceptions.**MissingTestParamKey**
Bases: *kissats.exceptions.KissATSError*

Required key missing in test parameter dictionary.

exception kissats.exceptions.**ObjectNotCallable**
Bases: *kissats.exceptions.KissATSError*

A callable object was expected

exception kissats.exceptions.**ResourceNotReady**
Bases: *kissats.exceptions.KissATSError*

Resource is not reserved or not ready.

exception kissats.exceptions.**ResourceRenewExceeded**
Bases: *kissats.exceptions.KissATSError*

Too many task reservation renews.

exception kissats.exceptions.**ResourceRetryExceeded**
Bases: *kissats.exceptions.KissATSError*

Too many task reservation retries.

exception kissats.exceptions.**ResourceUnavailable**
Bases: *kissats.exceptions.KissATSError*

Unable to reserve requested resource.

exception kissats.exceptions.**SchemaMisMatch**
Bases: *kissats.exceptions.KissATSError*

Something didn't match the specified schema

exception kissats.exceptions.**ServerCommandMissing**
Bases: *kissats.exceptions.KissATSError*

Server command missing in server request.

exception kissats.exceptions.**TaskPackageNotRegistered**
Bases: *kissats.exceptions.KissATSError*

A method or function was attempted that requires a valid task package to be registered.

exception kissats.exceptions.**UnsupportedRunMode**
Bases: *kissats.exceptions.KissATSError*

An unsupported run mode was requested

CHAPTER 13

Todo

Todo:

- Add quick start guide
- Add schema check to reporting/result return
- Update Task to use schemas
- Complete documentation
- Create sample ATS resource manager (server)
- Create sample test package
- Create sample test sequence optimizer
- Finish adding type hints
- Implement additional resource modes
- Add more logging
- Break up task_pack?
- DRY

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

R

S

T

U

V