sfini Documentation

Release 0.1.0

Laurie Opperman

Jun 23, 2019

Contents:

Create, run and manage AWS Step Functions easily. Pronounced "SFIN-ee".

This package aims to provide a user-friendly interface into defining and running Step Functions. Things you can do in `sfini` to interact with AWS Step Functions:

- Implement and register activities
- Define and register state machines
- Start, track and stop executions
- Run workers for activities
- Get information for registered activities and state machines
- De-register state machines and activities

Note: this is not a tool to convert Python code into a Step Functions state machine. For that, see pyawssfn.

CHAPTER 1

Installation

```
pip install sfini
```

CHAPTER 2

Documentation

2.1 sfini package

2.1.1 Subpackages and submodules

sfini.execution package

Submodules

sfini.execution.history module

State-machine execution history events.

Use `sfini.execution.Execution.format_history` for nice history printing.

class sfini.execution.history.**ActivityScheduled**(*timestamp*, *event_type*, *event_id*, *resource*, *previous_event_id=None*, *task_input=DefaultParameter()*, *timeout=None*, *heartbeat: int = None*)

Bases: *sfini.execution.history.LambdaFunctionScheduled*

An execution history activity task-schedule event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **resource** – AWS Lambda function ARN
- **previous_event_id** – identifying index of causal event
- **task_input** – task input

- **timeout** – time-out (seconds) of task execution
- **heartbeat** – heartbeat time-out (seconds)

class sfini.execution.history.**ActivityStarted**(*timestamp*, *event_type*, *event_id*, *worker_name: str*, *previous_event_id=None*)

Bases: *sfini.execution.history.Event*

An execution history activity task-start event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **worker_name** – name of activity worker executing activity task
- **previous_event_id** – identifying index of causal event

details_str

Format the event details.

Returns event details, formatted as string

Return type str

class sfini.execution.history.**Event**(*timestamp*, *event_type: str*, *event_id: int*, *previous_event_id: int = None*)

Bases: object

An execution history event.

Parameters

- **timestamp** (*datetime.datetime*) – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **previous_event_id** – identifying index of causal event

details_str

Format the event details.

Returns event details, formatted as string

Return type str

classmethod from_history_event(*history_event: Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]*) → Event

Parse an history event.

Parameters **history_event** – execution history event date, provided by AWS API

Returns constructed execution history event

Return type *Event*

class sfini.execution.history.**ExecutionStarted**(*timestamp, event_type, event_id, previous_event_id=None, execution_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter(), role_arn: str = None*)

Bases: *sfini.execution.history.Event*

An execution history execution-start event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **previous_event_id** – identifying index of causal event
- **execution_input** – execution input
- **role_arn** – execution AWS IAM role ARN

class sfini.execution.history.**Failed**(*timestamp, event_type, event_id, previous_event_id=None, error: str = None, cause: str = None*)

Bases: *sfini.execution.history.Event*

An execution history failure event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **previous_event_id** – identifying index of causal event
- **error** – error type
- **cause** – failure details

details_str

Format the event details.

Returns event details, formatted as string

Return type str

class sfini.execution.history.**LambdaFunctionScheduled**(*timestamp, event_type, event_id, resource: str, previous_event_id=None, task_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter(), timeout: int = None*)

Bases: *sfini.execution.history.Event*

An execution history AWS Lambda task-schedule event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **resource** – AWS Lambda function ARN
- **previous_event_id** – identifying index of causal event
- **task_input** – task input
- **timeout** – time-out (seconds) of task execution

details_str
Format the event details.

Returns event details, formatted as string

Return type str

class sfini.execution.history.**ObjectSucceeded**(*timestamp, event_type, event_id, previous_event_id=None, output: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter()*)

Bases: *sfini.execution.history.Event*

An execution history succeed event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **previous_event_id** – identifying index of causal event
- **output** – output of state/execution

class sfini.execution.history.**StateEntered**(*timestamp, event_type, event_id, state_name: str, previous_event_id=None, state_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter()*)

Bases: *sfini.execution.history.Event*

An execution history state-enter event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **state_name** – state name
- **previous_event_id** – identifying index of causal event
- **state_input** – state input

details_str
Format the event details.

Returns event details, formatted as string

Return type str

class sfini.execution.history.**StateExited**(*timestamp, event_type, event_id, state_name: str, previous_event_id=None, output: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter()*)

Bases: *sfini.execution.history.Event*

An execution history state-exit event.

Parameters

- **timestamp** – event time-stamp
- **event_type** – type of event
- **event_id** – identifying index of event
- **state_name** – state name
- **previous_event_id** – identifying index of causal event
- **output** – state output

details_str

Format the event details.

Returns event details, formatted as string

Return type str

sfini.execution.history.**parse_history**(*history_events: List[Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]]*) → List[sfini.execution.history.Event]

List the execution history.

Parameters **history_events** – history events as provided by AWS API

Returns history of execution events

Module contents

State-machine execution interfacing.

Executions track state-machine execution history, input, status and (if available) output. You can wait on it to finish, and iterate over its history.

class sfini.execution.**Execution**(*name: str, state_machine_arn: str, execution_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter(), arn: str = None, *, session: sfini._util.AWSSession = None*)

Bases: object

A state-machine execution.

Parameters

- **name** – name of execution
- **state_machine_arn** – executed state-machine ARN
- **execution_input** – execution input (must be JSON-serialisable)

- **arn** – execution ARN (if known: provided when execution is posted to AWS SFN)
- **session** – session to use for AWS communication

format_history() → str
Format the execution history for printing.

Returns history formatted

classmethod from_arn(*arn: str*, *, *session: sfini._util.AWSSession = None*) → sfini.execution._execution.Execution
Construct an `Execution` from an existing execution.

Queries AWS Step Functions for the execution with the given ARN

Parameters

- **arn** – existing execution ARN
- **session** – session to use for AWS communication

Returns described execution

classmethod from_execution_list_item(*item: Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]*, *, *session: sfini._util.AWSSession = None*) → Execution
Construct an `Execution` from a response list-item.

Parameters

- **item** – execution list item
- **session** – session to use for AWS communication

Returns described execution

get_history() → List[sfini.execution.history.Event]
List the execution history.

Returns history of execution events

output
Output of execution.

Raises `RuntimeError` – if execution is not yet finished, or execution failed

start()
Start this state-machine execution.

Sets the `arn` attribute.

start_date
Execution start time.

status
Execution status.

stop(*error_code: str = DefaultParameter()*, *details: str = DefaultParameter()*)
Stop an existing execution.

Parameters

- **error_code** – stop reason identification
- **details** – stop reason

Raises `RuntimeError` – if execution is already finished

stop_date
Execution stop time.

Raises RuntimeError – if execution is not yet finished

wait(*raise_on_failure: bool = True*, *timeout: float = None*)
Wait for execution to finish.

Parameters

- **raise_on_failure** – raise error when execution fails
- **timeout** – time to wait for execution to finish (seconds), default: no time-out

Raises RuntimeError – if execution fails, or if time-out is reached before execution finishes

sfini.state package

Submodules

sfini.state.choice module

SFN choice rules.

These rules are used in the 'Choice' state of a state-machine, and allow for conditional branching in the state-machine. There are two types of choice rule: comparisons and logical operations.

class sfini.state.choice.**And**(*choice_rules: List[sfini.state.choice.ChoiceRule]*, *next_state=None*)
Bases: sfini.state.choice._NonUnary

class sfini.state.choice.**BooleanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**ChoiceRule**(*next_state=None*)
Bases: object

A choice case for the 'Choice' state.

Parameters **next_state** (sfini.state.State) – state to execute on success

to_dict() → Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]
Convert this rule to a definition dictionary.

Returns definition

class sfini.state.choice.**Comparison**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.ChoiceRule*

Compare variable value.

Parameters

- **variable_path** – path of variable to compare
- **comparison_value** – value to compare against
- **next_state** – state to execute on success

to_dict()
Convert this rule to a definition dictionary.

Returns definition

class sfini.state.choice.**Logical**(*next_state=None*)
Bases: *sfini.state.choice.ChoiceRule*

class sfini.state.choice.**Not**(*choice_rule: sfini.state.choice.ChoiceRule*, *next_state=None*)
Bases: *sfini.state.choice.Logical*

Logical 'not' operation on a choice rule.

Parameters

- **choice_rule** – choice rule to operate on
- **next_state** – state to execute on success

class sfini.state.choice.**NumericEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**NumericGreaterThan**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**NumericGreaterThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**NumericLessThan**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**NumericLessThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**Or**(*choice_rules: List[sfini.state.choice.ChoiceRule]*, *next_state=None*)
Bases: sfini.state.choice._NonUnary

class sfini.state.choice.**StringEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**StringGreaterThan**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**StringGreaterThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**StringLessThan**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**StringLessThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: *sfini.state.choice.Comparison*

class sfini.state.choice.**TimestampEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: sfini.state.choice._TimestampRule

class sfini.state.choice.**TimestampGreaterThan**(*variable_path: str*, *comparison_value*, *next_state=None*)
Bases: sfini.state.choice._TimestampRule

class sfini.state.choice.**TimestampGreaterThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)

Bases: sfini.state.choice._TimestampRule

class sfini.state.choice.**TimestampLessThan**(*variable_path: str*, *comparison_value*, *next_state=None*)

Bases: sfini.state.choice._TimestampRule

class sfini.state.choice.**TimestampLessThanEquals**(*variable_path: str*, *comparison_value*, *next_state=None*)

Bases: sfini.state.choice._TimestampRule

Module contents

State definitions.

States comprise a state-machine, defining its logic and which activities to run, and direct data.

class sfini.state.**State**(*name: str*, *comment: str = DefaultParameter()*, *input_path: Optional[str] = DefaultParameter()*, *output_path: Optional[str] = DefaultParameter()*)

Bases: object

Abstract state.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

add_to(*states*)

Add this state to a state-machine definition.

Any child states will also be added to the definition.

Parameters **states** (*dict[str, State]*) – state-machine states

to_dict() → Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]

Convert this state to a definition dictionary.

Returns definition

class sfini.state.**HasNext**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*)

Bases: sfini.state._base.State

State able to advance mix-in.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

next

next state to execute, or None if state is terminal

add_to(*states*)
Add this state to a state-machine definition.

Any child states will also be added to the definition.

Parameters **states** (*dict[str, State]*) – state-machine states

goes_to(*state: sfini.state._base.State*)
Set next state after this state finishes.

Parameters **state** – state to execute next

remove_next()
Remove next state, making this state terminal.

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**HasResultPath**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*, *result_path: Optional[str] = DefaultParameter()*)
Bases: sfini.state._base.State

State with result mix-in.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output
- **result_path** – task output location JSONPath, None for discarded output

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**CanRetry**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*)
Bases: sfini.state._base.State

Retryable state mix-in.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

retriers
error handler policies

retry_for(*errors: Sequence[str], interval: int = DefaultParameter(), max_attempts: int = DefaultParameter(), backoff_rate: float = DefaultParameter()*)
Add a retry handler.

Parameters

- **errors** – codes of errors for retry to be executed. See AWS Step Functions documentation
- **interval** – (initial) retry interval (seconds)
- **max_attempts** – maximum number of attempts before re-raising error
- **backoff_rate** – retry interval increase factor between attempts

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**CanCatch**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*)
Bases: sfini.state._base.State

Exception catching state mix-in.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

catchers
error handler policies

add_to(*states*)
Add this state to a state-machine definition.

Any child states will also be added to the definition.

Parameters **states** (*dict[str, State]*) – state-machine states

catch(*errors: Sequence[str], next_state: sfini.state._base.State, result_path: Optional[str] = DefaultParameter()*)
Add an error handler.

Parameters

- **errors** – code of errors for catch clause to be executed. See AWS Step Functions documentation
- **next_state** – state to execute for catch clause
- **result_path** – error details location JSONPath

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Succeed**(*name: str*, *comment: str = DefaultParameter()*, *input_path: Optional[str] = DefaultParameter()*, *output_path: Optional[str] = DefaultParameter()*)
Bases: sfini.state._base.State

End execution successfully.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, `None` for empty input
- **output_path** – state output filter JSONPath, `None` for discarded output

class sfini.state.**Fail**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*, *cause: str = DefaultParameter()*, *error: str = DefaultParameter()*)

Bases: `sfini.state._base.State`

End execution unsuccessfully.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, `None` for empty input
- **output_path** – state output filter JSONPath, `None` for discarded output
- **error** – error type
- **cause** – failure description

to_dict()

Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Pass**(*name, comment=DefaultParameter(), input_path=DefaultParameter(), output_path=DefaultParameter(), result_path=DefaultParameter(), result: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]] = DefaultParameter()*)

Bases: `sfini.state._base.HasResultPath`, `sfini.state._base.HasNext`, `sfini.state._base.State`

No-op state, possibly introducing data.

The name specifies the location of any introduced data.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, `None` for empty input
- **output_path** – state output filter JSONPath, `None` for discarded output
- **result_path** – task output location JSONPath, `None` for discarded output
- **result** – return value of state, stored in the variable `name`

next

next state to execute

to_dict()

Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Wait**(*name, until: Union[int, datetime.datetime, str], comment=DefaultParameter(), input_path=DefaultParameter(), output_path=DefaultParameter()*)

Bases: sfini.state._base.HasNext, sfini.state._base.State

Wait for a time before continuing.

Parameters

- **name** – name of state
- **until** – time to wait. If int, then seconds to wait; if datetime.datetime, then time to wait until; if str, then name of state-variable containing time to wait until
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

next
next state to execute

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Parallel**(*name, comment=DefaultParameter(), input_path=DefaultParameter(), output_path=DefaultParameter(), result_path=DefaultParameter()*)

Bases: sfini.state._base.HasResultPath, sfini.state._base.HasNext, sfini.state._base.CanRetry, sfini.state._base.CanCatch, sfini.state._base.State

Run states-machines in parallel.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output
- **result_path** – task output location JSONPath, None for discarded output

branches
state-machines to run in parallel. These state-machines do not need to be registered with AWS Step Functions.

Type list[sfini.StateMachine]

next
next state to execute

retriers
retry conditions

catchers
handled state errors

add(*state_machine*)
Add a state-machine to be executed.

The input to the state-machine execution is the input into this parallel state. The output of the parallel state is a list of each state-machine's output (in order of adding).

Parameters **state_machine** (*sfini.StateMachine*) – state-machine to add. It will be run when this task is executed. Added state-machines do not need to be registered with AWS Step Functions

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Choice**(*name*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*)
Bases: sfini.state._base.State

Branch execution based on comparisons.

Parameters

- **name** – name of state
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output

choices
choice rules determining branch conditions

Type list[sfini.choice.ChoiceRule]

default
fall-back state if all comparisons fail, or None for no fall-back (Step Functions will raise a 'States.NoChoiceMatched' error)

add(*rule*)
Add a choice-rule.

Parameters **rule** (*sfini.choice.ChoiceRule*) – branch execution condition and specification to add

Raises RuntimeError – rule doesn't specify next-state

add_to(*states*)
Add this state to a state-machine definition.

Any child states will also be added to the definition.

Parameters **states** (*dict[str, State]*) – state-machine states

remove(*rule*)
Remove a branch.

Parameters **rule** (*sfini.choice.ChoiceRule*) – branch execution condition and specification to remove

Raises ValueError – if rule is not a registered branch

set_default(*state: sfini.state._base.State*)
Set the default state to execute when no conditions were met.

Parameters **state** – default state to execute

to_dict()
Convert this state to a definition dictionary.

Returns definition

class sfini.state.**Task**(*name*, *resource*, *comment=DefaultParameter()*, *input_path=DefaultParameter()*, *output_path=DefaultParameter()*, *result_path=DefaultParameter()*, *timeout: int = DefaultParameter()*)

Bases: sfini.state._base.HasResultPath, sfini.state._base.HasNext, sfini.state._base.CanRetry, sfini.state._base.CanCatch, sfini.state._base.State

Activity execution.

Parameters

- **name** – name of state
- **resource** (sfini.task_resource.TaskResource) – task executor, eg activity or Lambda function
- **comment** – state description
- **input_path** – state input filter JSONPath, None for empty input
- **output_path** – state output filter JSONPath, None for discarded output
- **result_path** – task output location JSONPath, None for discarded output
- **timeout** – seconds before task time-out

next
next state to execute

retriers
retry conditions

catchers
handled state errors

to_dict()
Convert this state to a definition dictionary.

Returns definition

sfini.activity module

Activity interfacing.

Activities are separate from state-machines, and are used as implementations of 'Task' states. Activities are registered separately.

class sfini.activity.**Activity**(*name: str*, ***, *session: sfini._util.AWSSession = None*)

Bases: *sfini.task_resource.TaskResource*

Activity execution.

Note that activity names must be unique (within a region). It's recommended to put your code's title and version in the activity name. Activities makes this straight-forward.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run some activities using their names.

Parameters

- **name** – name of activity
- **session** – session to use for AWS communication

deregister()
Remove activity from AWS SFN.

is_registered() → bool
See if this activity is registered with AWS SFN.

Returns if this activity is registered

register()
Register activity with AWS SFN.

service = 'activity'

class sfini.activity.**ActivityRegistration**(*prefix: str = ''*, ***, *session: sfini._util.AWSSession = None*)

Bases: object

Activities registration.

Provides convenience for grouping activities, generating activity names, bulk-registering activities, and activity function decoration.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run an activity using the registered activity name.

Parameters

- **prefix** – prefix for activity names
- **session** – session to use for AWS communication

activities
registered activities

Example

```
>>> activities = ActivityRegistration(prefix="foo")
>>> @activities.activity(name="MyActivity")
>>> def fn(data):
...     print("hi")
>>> print(fn.name)
fooMyActivity
```

activity(*name: str = None*, *heartbeat: int = 20*) → Callable[[Callable], sfini.activity.CallableActivity]
Activity function decorator.

The decorated function will be passed one argument: the input to the task state that executes the activity.

Parameters

- **name** – name of activity, default: function name
- **heartbeat** – seconds between heartbeat during activity running

add_activity(*activity: sfini.activity.Activity*)
Add an activity to the group.

Parameters **activity** – activity to add

Raises ValueError – if activity name already in-use in group

deregister()
Remove activities in AWS SFN.

register()
Add registered activities to AWS SFN.

smart_activity(*name: str = None*, *heartbeat: int = 20*) → Callable[[Callable], sfini.activity.SmartCallableActivity]
Smart activity function decorator.

The decorated function will be passed values to its parameters from the input to the task state that executes the activity.

Parameters

- **name** – name of activity, default: function name
- **heartbeat** – seconds between heartbeat during activity running

class sfini.activity.**CallableActivity**(*name*, *fn: Callable*, *heartbeat=20*, ***, *session=None*)
Bases: *sfini.activity.Activity*

Activity execution defined by a callable.

Note that activity names must be unique (within a region). It's recommended to put your application's name and version in the activity name. `ActivityRegistration` makes this straight-forward.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run an activity using the registered activity name.

Parameters

- **name** – name of activity
- **fn** – function to run activity
- **heartbeat** – seconds between heartbeat during activity running
- **session** – session to use for AWS communication

call_with(*task_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]*) → Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]
Call with task-input context.

Parameters **task_input** – task input

Returns function return-value

classmethod decorate(*name: str*, *heartbeat: int = 20*, ***, *session: sfini._util.AWSSession = None*) → Callable[[Callable], sfini.activity.CallableActivity]
Decorate a callable as an activity implementation.

Parameters

- **name** – name of activity
- **heartbeat** – seconds between heartbeat during activity running
- **session** – session to use for AWS communication

class sfini.activity.**SmartCallableActivity**(*name*, *fn: Callable*, *heartbeat=20*, ***, *session=None*)
Bases: *sfini.activity.CallableActivity*

Activity execution defined by a callable, processing input.

The arguments to `fn` are extracted from the input provided by AWS Step Functions.

Note that activity names must be unique (within a region). It's recommended to put your application's name and version in the activity name. `ActivityRegistration` makes this straight-forward.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run an activity using the registered activity name.

Parameters

- **name** – name of activity
- **fn** – function to run activity
- **heartbeat** – seconds between heartbeat during activity running
- **session** – session to use for AWS communication

sig
function signature

call_with(*task_input: Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]*)
Call with task-input context.

Parameters **task_input** – task input

Returns function return-value

sfini.state_machine module

State-machine interfacing.

A state-machine defines the logic for a workflow of an application. It is comprised of states (ie stages), and executions of which will run the workflow over some given data.

class sfini.state_machine.**StateMachine**(*name: str, states: Dict[str, sfini.state._base.State], start_state: str, comment: str = DefaultParameter(), timeout: int = DefaultParameter(), *, session: sfini._util.AWSSession = None*)
Bases: object

State machine structure for AWS Step Functions.

Parameters

- **name** – name of state-machine
- **states** – state-machine states
- **start_state** – name of start state
- **comment** – description of state-maching
- **timeout** – execution time-out (seconds)
- **session** – session to use for AWS communication

arn
State-machine generated ARN.

default_role_arn
sfini-generated state-machine IAM role ARN.

deregister()
Remove state-machine from AWS SFN.

is_registered() → bool
See if this state-machine is registered with AWS SFN.

Returns if this state-machine is registered

list_executions(*status: str = None*) → List[sfini.execution._execution.Execution]
List all executions of this state-machine.

This state-machine is manually attached to the `state_machine` attribute of the resultant executions here.

Parameters **status** – only list executions with this status. Choose from 'RUNNING', 'SUCCEEDED', 'FAILED', 'TIMED_OUT' or 'ABORTED'

Returns executions of this state-machine

register(*role_arn: str = None*, *allow_update: bool = False*)
Register state-machine with AWS SFN.

Parameters

- **role_arn** – state-machine IAM role ARN
- **allow_update** – allow overwriting of an existing state-machine with the same name

start_execution(*execution_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]*) → sfini.execution._execution.Execution
Start an execution.

Parameters **execution_input** – input to first state in state-machine

Returns started execution

to_dict() → Dict[str, Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]]
Convert this state-machine to a definition dictionary.

Returns definition

sfini.state_machine.**construct_state_machine**(*name: str*, *start_state: sfini.state._base.State*, *comment: str = DefaultParameter()*, *timeout: int = DefaultParameter()*, ***, *session: sfini._util.AWSSession = None*) → sfini.state_machine.StateMachine
Construct a state-machine from the starting state.

Make sure to construct the state-machine after all states have been defined: subsequent states will need to be added to the state-machine manually.

Only states referenced by the provided first state (and their children) will be in the state-machine definition. Add states via an impossible choice rule to include them in the definition.

Parameters

- **name** – name of state-machine
- **start_state** – starting state of state-machine
- **comment** – description of state-maching
- **timeout** – execution time-out (seconds)
- **session** – session to use for AWS communication

Returns constructed state-machine

sfini.task_resource module

Task resource interfacing.

'Task' states require some executor to implement the task, which different AWS services can provide, including Step Functions activities and Lambda functions.

class sfini.task_resource.**Lambda**(*name: str*, *, *session: sfini._util.AWSSession = None*)

Bases: *sfini.task_resource.TaskResource*

AWS Lambda function executor for a task.

Parameters

- **name** – name of Lambda function
- **session** – session to use for AWS communication

arn

Task resource generated ARN.

service = 'function'

class sfini.task_resource.**TaskResource**(*name: str*, *, *session: sfini._util.AWSSession = None*)

Bases: object

Task execution.

An instance of this represents a service which can run tasks defined in a state-machine.

Parameters

- **name** – name of resource
- **session** – session to use for AWS communication

service

resource type

arn

Task resource generated ARN.

service = None

sfini.worker module

Activity task polling and execution.

You can provide you're own workers: the interface to the activities is public. This module's worker implementation uses threading, and is designed to be resource-managed outside of Python.

class sfini.worker.**TaskExecution**(*activity*, *task_token: str*, *task_input: Union[None, bool, str, int, float, List[JSONable], Dict[str, JSONable]]*, *, *session: sfini._util.AWSSession = None*)

Bases: object

Execute a task, providing heartbeats and catching failures.

Parameters

- **activity** (sfini.activity.CallableActivity) – activity to execute task of
- **task_token** – task token for execution identification
- **task_input** – task input
- **session** – session to use for AWS communication

report_cancelled()
Cancel a task execution: stop interaction with SFN.

run()
Run task.

class sfini.worker.**Worker**(*activity*, *name: str = None*, *, *session: sfini._util.AWSSession = None*)
Bases: object

Worker to poll for activity task executions.

Parameters

- **activity** (sfini.activity.CallableActivity) – activity to poll and run executions of
- **name** – name of worker, used for identification, default: a combination of UUID and host's FQDN
- **session** – session to use for AWS communication

end()
End polling.

join()
Block until polling exit.

run()
Run worker to poll for and execute specified tasks.

start()
Start polling.

exception sfini.worker.**WorkerCancel**(**args*, ***kwargs*)
Bases: KeyboardInterrupt

Workflow execution interrupted by user.

2.1.2 Module contents

AWS Step Functions service.

class sfini.**AWSSession**(*session: boto3.session.Session = None*)
Bases: object

AWS session, for preconfigure communication with AWS.

Parameters **session** – session to use

account_id
Session's account's account ID.

credentials
AWS session credentials.

region
Session AWS region.

sfn
Step Functions client.

class sfini.**Activity**(*name: str*, *, *session: sfini._util.AWSSession = None*)
Bases: *sfini.task_resource.TaskResource*

Activity execution.

Note that activity names must be unique (within a region). It's recommended to put your code's title and version in the activity name. `Activities` makes this straight-forward.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run some activities using their names.

Parameters

- **name** – name of activity
- **session** – session to use for AWS communication

deregister()
Remove activity from AWS SFN.

is_registered() → bool
See if this activity is registered with AWS SFN.

Returns if this activity is registered

register()
Register activity with AWS SFN.

service = 'activity'

class sfini.**ActivityRegistration**(*prefix: str = ''*, ***, *session: sfini._util.AWSSession = None*)
Bases: `object`

Activities registration.

Provides convenience for grouping activities, generating activity names, bulk-registering activities, and activity function decoration.

An activity is attached to state-machine tasks, and is called when that task is executed. A worker registers itself able to run an activity using the registered activity name.

Parameters

- **prefix** – prefix for activity names
- **session** – session to use for AWS communication

activities
registered activities

Example

```
>>> activities = ActivityRegistration(prefix="foo")
>>> @activities.activity(name="MyActivity")
>>> def fn(data):
...     print("hi")
>>> print(fn.name)
fooMyActivity
```

activity(*name: str = None*, *heartbeat: int = 20*) → Callable[[Callable], sfini.activity.CallableActivity]
Activity function decorator.

The decorated function will be passed one argument: the input to the task state that executes the activity.

Parameters

- **name** – name of activity, default: function name
- **heartbeat** – seconds between heartbeat during activity running

add_activity(*activity: sfini.activity.Activity*)
Add an activity to the group.

Parameters **activity** – activity to add

Raises `ValueError` – if activity name already in-use in group

deregister()
Remove activities in AWS SFN.

register()
Add registered activities to AWS SFN.

smart_activity(*name: str = None*, *heartbeat: int = 20*) → Callable[[Callable], sfini.activity.SmartCallableActivity]
Smart activity function decorator.

The decorated function will be passed values to its parameters from the input to the task state that executes the activity.

Parameters

- **name** – name of activity, default: function name
- **heartbeat** – seconds between heartbeat during activity running

class sfini.**CLI**(*state_machine=None*, *activities=None*, *role_arn: str = None*, *version: str = None*, *prog: str = None*)
Bases: `object`

`sfini` command-line interface.

Parameters

- **state_machine** (*sfini.StateMachine*) – state-machine interact with
- **activities** (`sfini.ActivityRegistration`) – activities to poll for
- **role_arn** – AWS ARN for state-machine IAM role
- **version** – version to display, default: no version display
- **prog** – program name displayed in program help, default: `sys.argv[0]`

parse_args()
Parse command-line arguments and run CLI.

class sfini.**Lambda**(*name: str*, ***, *session: sfini._util.AWSSession = None*)
Bases: *sfini.task_resource.TaskResource*

AWS Lambda function executor for a task.

Parameters

- **name** – name of Lambda function
- **session** – session to use for AWS communication

arn
Task resource generated ARN.

service = 'function'

sfini.**construct_state_machine**(*name: str*, *start_state: sfini.state._base.State*, *comment: str = DefaultParameter()*, *timeout: int = DefaultParameter()*, ***, *session: sfini._util.AWSSession = None*) → sfini.state_machine.StateMachine

Construct a state-machine from the starting state.

Make sure to construct the state-machine after all states have been defined: subsequent states will need to be added to the state-machine manually.

Only states referenced by the provided first state (and their children) will be in the state-machine definition. Add states via an impossible choice rule to include them in the definition.

Parameters

- **name** – name of state-machine
- **start_state** – starting state of state-machine
- **comment** – description of state-maching
- **timeout** – execution time-out (seconds)
- **session** – session to use for AWS communication

Returns constructed state-machine

class sfini.**Worker**(*activity*, *name: str = None*, ***, *session: sfini._util.AWSSession = None*)

Bases: object

Worker to poll for activity task executions.

Parameters

- **activity** (sfini.activity.CallableActivity) – activity to poll and run executions of
- **name** – name of worker, used for identification, default: a combination of UUID and host's FQDN
- **session** – session to use for AWS communication

end()

End polling.

join()

Block until polling exit.

run()

Run worker to poll for and execute specified tasks.

start()

Start polling.

exception sfini.**WorkerCancel**(**args*, ***kwargs*)

Bases: KeyboardInterrupt

Workflow execution interrupted by user.

2.2 Examples

More examples:

- *File-processing*

- *Looping*
- *Parallel*
- *CLI*
- *Error-handling*

2.2.1 My first `sfini`

First, a step-by-step example. We'll begin by defining activities:

```
import sfini

activities = sfini.ActivityRegistration(prefix="test")


@activities.activity(name="addActivity")
def add_activity(data):
    return data["a"] + data["b"]
```

We've created one activity, which when passed some data, will add two of the values in that data and return the result. This activity is independent of any state-machine, and will always do what we define it to do. We're using a prefix in activities registeration to help with unregistering later.

Next, let's define a simple state-machine to utilise our adding activity:

```
add = sfini.Task("add", add_activity)
sm = sfini.construct_state_machine("testAdding", add)
```

We've added a 'task' as the initial (and in this example, only) state (ie stage) of the workflow. This task will be implemented by our adding activity. The workflow input always gets passed to its first state, and in this example we are passing all of the state input into the activity (same for the output: all activity output goes to the state, which becomes the workflow output).

To be able to use this activity and state-machine, we must register it with AWS Step Functions:

```
activities.register()
sm.register()
```

You may need to pass a role ARN for an IAM account which has permissions to run state-machine executions: call `sm.register(role_arn="...")`.

Now, let's start an execution of the state-machine, with some input:

```
execution = sm.start_execution(execution_input={"a": 3, "b": 42})
print(execution.name)
# testAdding_2019-05-13T19-07_0354d790
```

The execution is now started, however it's blocked on the 'add' task (which is the only task). We've now declared, defined and registered our adding activity, but we need a worker to be able to run executions of the activity. *sfini*'s workers are implemented in threads, but you're welcome to bring your own

Start a worker to allow the workflow execution to progress through the 'add' task:

```
worker = sfini.Worker(add_activity)
worker.start()
```

We can now block the local script's execution by waiting for the execution to finish:

```
execution.wait()
print(execution.output)
# 45
```

Executions track the progress of the running of the state-machine, and have knowledge of the full history of the process. Once they're finished, we can get the workflow's output, like above.

Clean-up: turn off our workers. Calling `end` on the worker prevents new activity executions from occuring, but won't kill any current executions (use CTRL+C or your favourite interrupt/kill signal sender for that). `join` simply waits for the thread to finish:

```
worker.end()
worker.join()
```

And more clean-up: unregister the adding activity and state-machine (unless you're felling particularly attached):

```
activities.deregister()
sm.deregister()
```

This will only unregister the adding activity.

2.2.2 More examples

Enabling log output for these examples may be helpful:

```
import logging as lg
lg.basicConfig(
    level=lg.DEBUG,
    format="[%(levelname)8s] %(name)s: %(message)s")
```

File-processing

```
import sfini
import pathlib
from PIL import Image

# Define activities
activities = sfini.ActivityRegistration(prefix="sfiniActs")


@activities.smart_activity("resizeActivity")
def resize_activity(image_dir, resized_image_dir, new_size=(64, 64)):
    image_dir = pathlib.Path(image_dir)
    resized_image_dir = pathlib.Path(resized_image_dir)
    for path in image_dir.iterdir():
        resized_path = resized_image_dir / path.relative_to(image_dir)
        print("Resizing image '%s'" % path)
        Image.open(path).resize(new_size).save(resized_path)


@activities.activity("getCentresActivity")
def get_centres_activity(resized_image_dir):
    resized_image_dir = pathlib.Path(resized_image_dir)
    centres = []
```

(continues on next page)

(continued from previous page)

```
    for path in resized_image_dir.iterdir():
        im = Image.open(path)
        centres.append(im.getpixel((im.size[0] // 2, im.size[1] // 2)))
    return centres


# Define state-machine
resize_images = sfini.Task(
    "resizeImages",
    resize_activity,
    result_path=None)

get_centres = sfini.Task(
    "getCentre",
    get_centres_activity,
    comment="get pixel values of centres of images",
    input_path="$.resized_image_dir",
    result_path="$.res")
resize_images.goes_to(get_centres)

sm = sfini.construct_state_machine("sfiniSM", resize_images)

# Register state-machine and activities
activities.register()
sm.register()

# Start activity workers
workers = [
    sfini.Worker(resize_activity),
    sfini.Worker(get_centres_activity)]
[w.start() for w in workers]

# Start execution
execution = sm.start_execution(
    execution_input={
        "image_dir": "~/data/images/",
        "resized_image_dir": "~/data/images-small/"})
print(execution.name)
# sfiniSM-07-11T19-07_0354d790

# Wait for execution and print output
execution.wait()
print(execution.output)
# {
#     "image_dir": "~/data/images/",
#     "resized_image_dir": "~/data/images-small/"
#     "res": [(128, 128, 128), (128, 255, 0), (0, 0, 0), (0, 0, 255)]}

# Stop activity workers
[w.end() for w in workers]
[w.join() for w in workers]

# Deregister state-machine and activities
activities.deregister()
sm.deregister()
```

Looping

```
import sfini

# Define activities
activities = sfini.ActivityRegistration(prefix="sfiniActs")


@activities.activity("increment")
def increment_activity(data):
    return data["counter"] + data["increment"]


# Define state-machine
initialise = sfini.Pass(
    "initialise",
    result=0,
    result_path="$.counter")

increment = sfini.Task(
    "increment",
    increment_activity,
    result_path="$.counter")
initialise.goes_to(increment)

check_counter = sfini.Choice("checkCounter")
increment.goes_to(check_counter)

check_counter.add(sfini.NumericLessThan("$.counter", 10, increment))

end = sfini.Succeed("end", output_path="$.counter")
check_counter.set_default(end)

sm = sfini.construct_state_machine("sfiniSM", initialise)

# Register state-machine and activities
activities.register()
sm.register()

# Start activity workers
worker = sfini.Worker(increment_activity)
worker.start()

# Start execution
execution = sm.start_execution(execution_input={"increment": 3})
print(execution.name)
# sfiniSM-07-11T19-07_0354d790

# Wait for execution and print output
execution.wait()
print(execution.output)
# 12

# Stop activity workers
worker.end()
worker.join()
```

(continues on next page)

(continued from previous page)

```
# Deregister state-machine and activities
activities.deregister()
sm.deregister()
```

Parallel

```
import sfini
import datetime
import logging as lg

# Define activities
activities = sfini.ActivityRegistration(prefix="sfiniActs")


@activities.activity("logActivity")
def log_message_activity(data):
    lg.log(data["level"], data["message"])


@activities.activity("printActivity")
def print_message_activity(message):
    print(message)
    diff = datetime.timedelta(seconds=len(message) * 5)
    now = datetime.datetime.now(tz=datetime.timezone.utc)
    return (now + diff).isoformat()


# Define state-machine
print_and_log = sfini.Parallel(
    "printAndLog",
    result_path="$.parallel",
    output_path="$.parallel")

log = sfini.Task("log", log_message_activity, result_path=None)
log_sm = sfini.construct_state_machine("logSM", log)

print_ = sfini.Task(
    "print",
    print_message_activity,
    result_path="$.until")
wait = sfini.Wait("wait", "$.until")
print_.goes_to(wait)
print_sm = sfini.construct_state_machine("printSM", print_)

print_and_log.add(log_sm)
print_and_log.add(print_sm)

sm = sfini.construct_state_machine("sfiniSM", print_and_log)

# Register state-machine and activities
activities.register()
sm.register()

# Start activity workers
workers = [
```

(continues on next page)

(continued from previous page)

```
    sfini.Worker(log_message_activity),
    sfini.Worker(print_message_activity)]
[w.start() for w in workers]

# Start execution
execution = sm.start_execution(
    execution_input={"level": 20, "message": "foo"})
print(execution.name)
# sfiniSM-07-11T19-07-26.53_0354d790

# Wait for execution and print output
execution.wait()
print(execution.output)
# [
#     {"level": 20, "message": "foo"},
#     {"level": 20, "message": "foo", "until": "2018-07-11T19-07-42.53"}]

# Stop activity workers
[w.end() for w in workers]
[w.join() for w in workers]

# Deregister state-machine and activities
activities.deregister()
sm.deregister()
```

CLI

```
import sfini

# Define activities
activities = sfini.ActivityRegistration(prefix="sfiniActs")


@activities.activity("printActivity")
def print_activity(data):
    print(data)


# Define state-machine
print_ = sfini.Task("print", print_activity)
sm = sfini.construct_state_machine("sfiniSM", print_)

# Parse arguments
sfini.CLI(sm, activities, role_arn="...", version="1.0").parse_args()
```

Error-handling

```
import sfini
import time

# Define activities
activities = sfini.ActivityRegistration(prefix="sfiniActs")
```

(continues on next page)

(continued from previous page)

```
sleep_time = 15


class MyError(Exception):
    pass


@activities.activity("raiseActivity")
def raise_activity(data):
    global sleep_time
    time.sleep(sleep_time)
    sleep_time -= 10
    raise MyError("foobar")


# Define state-machine
raise_ = sfini.Task("raise", raise_activity, timeout=10)
raise_.retry_for(["States.Timeout"], interval=3)

fail = sfini.Fail(
    "fail",
    error="WorkerError",
    cause="MyError was raised")
raise_.catch(["MyError"], fail, result_path="$.error-info")

sm = sfini.construct_state_machine("sfiniSM", raise_)

# Register state-machine and activities
activities.register()
sm.register()

# Start activity workers
worker = sfini.Worker(raise_activity)
worker.start()

# Start execution
execution = sm.start_execution(execution_input={})
print(execution.name)
# sfiniSM-07-11T19-07_0354d790

# Wait for execution and print output
execution.wait()
print(execution.format_history())
# ExecutionStarted [1] @ 2019-06-23 19:27:34.026000+10:00
# TaskStateEntered [2] @ 2019-06-23 19:27:34.052000+10:00:
#   name: raise
# ActivityScheduled [3] @ 2019-06-23 19:27:34.052000+10:00:
#   resource: arn:...:sfiniActsraiseActivity
# ActivityStarted [4] @ 2019-06-23 19:27:34.130000+10:00:
#   worker: myWorker-81a5a3e4
# ActivityTimedOut [5] @ 2019-06-23 19:27:44.131000+10:00:
#   error: States.Timeout
# ActivityScheduled [6] @ 2019-06-23 19:27:47.132000+10:00:
#   resource: arn:...:sfiniActsraiseActivity
# ActivityStarted [7] @ 2019-06-23 19:30:45.637000+10:00:
#   worker: myWorker-4b6b9dfb
# ActivityFailed [8] @ 2019-06-23 19:30:50.908000+10:00:
```

(continues on next page)

(continued from previous page)

```
#   error: MyError
# TaskStateExited [9] @ 2019-06-23 19:30:50.908000+10:00:
#   name: raise
# FailStateEntered [10] @ 2019-06-23 19:30:50.916000+10:00:
#   name: fail
# ExecutionFailed [11] @ 2019-06-23 19:30:50.916000+10:00:
#   error: WorkerError

# Stop activity workers
worker.end()
worker.join()

# Deregister state-machine and activities
activities.deregister()
sm.deregister()
```

CHAPTER 3

Indices and tables

- genindex

Python Module Index

S

Index

T

W