# rhodes

*Release 0.5.4*

Jan 14, 2020

# Project Operations

Rhodes is a library designed to make creating AWS Step Functions state machines simple and less error prone. Rhodes is designed to integrate tightly with Troposphere to provide a seamless experience for building state machines in your AWS CloudFormation templates.

For some examples of what this looks like, check out the examples.

---

**Important:** This project is a work in progress and is not ready for production use. The low-level API is unlikely to change, but the higher-level helper APIs might as I find better patterns.

---

CHAPTER 1

# Versioning

Until v1.0.0, rhodes will adhere to the following versioning policy:

Given a version number 0.MINOR.PATCH, the:

- MINOR version will increment for any feature additions and MAY contain breaking changes.
- PATCH version will increment for minor changes and bugfixes.

After v1.0.0, rhodes releases will follow Semantic Versioning.

CHAPTER 2

# Contributing

## 2.1 Where to Start?

Want to help but you're not sure where to start? Check our issues for `help wanted` or `good first issue` labels.

Nothing there? Examples and documentation are always a great place to start! If you're confused about something, chances are a lot of other people are too. If you can offer a clearer way of explaining something or an example that highlights what you were trying to figure out, that's a great way to contribute!

### 2.1.1 Development Workflow

All development should start and end with the `development` branch. We create all releases from the `master` branch, but no human should ever touch that branch.

If you forget, don't worry! `master` should never be far behind `development` so you probably won't have any conflicts if you start from `master` and our army of bots will make sure that your pull requests end up in the right place. :)

## 2.2 Development Environment

All of our checks and tests can be run locally using `tox`. Just `pipx install tox` or `pip install --user -r ci-requirements.txt` to get set up.

Once you have `tox` installed, check out the config file for information on the various test environments.

For more information on `tox`, see the tox docs.

## 2.3 Submitting Changes

When you're ready to submit your changes, open up a pull request against the `development` branch.

## 2.4 Operations

Ok, so that's cool and all, but how does all of that work???

We use a variety of GitHub Apps and Action workflows to manage this repo:

### 2.4.1 Repo Management

- settings : We use probot settings to manage most repository settings.
- branch-switcher : We use probot branch switcher to make sure that all pull requests end up with the correct target.
- apply-pr-labels Some of our automation relies on labels being applied to pull requests. This workflow takes care of applying those labels.
- promote Every time a pull request is merged into `development`, this workflow automatically creates a pull request promoting those changes to `master`. If a pull request already exists, it will simply be updated as you would expect.
- automerge Our branch protection rules define our requirements for a pull request to be accepted. If those requirements are met, this workflow merges those changes.
- publish Libraries are not terribly useful if they are not published to the appropriate package indexes. This workflow takes any published GitHub releases, packages them, and publishes them to PyPI.

### 2.4.2 Pull Requests and Testing

- static-analysis : We check all pull requests with a variety of static analysis tools to ensure the safety, consistency, and quality of our codebase. This also means that computers (not humans!) will complain if you don't have formatting/etc quite right. :)
- local-tests : We run our local (unit and functional) tests on every pull request.
- integration-tests : Because they require credentials that are difficult to share, we only run our integration tests on promotion pull requests from `development` to `master`. If you want to run these tests for yourself locally, though, you can! Like all of our other tests, our integration tests are all managed through `tox`.

CHAPTER 3

# Changelog

## 3.1 0.5.4 – 2020-01-01

- Minor cleanup.
- Lots of docs updates.

## 3.2 0.5.3 – 2019-12-16

### 3.2.1 bugfixes

- Fix `Pass` type stub to include `Result`. #61
- Fix `ContextPath` to allow indexing inside of `Map.Item.Value`. #62

## 3.3 0.5.2 – 2019-12-10

- Fixed incorrect kw-only split for `AwsStepFunctions` type signature. #55
- Fixed too-strict type restrictions. #54 #56

## 3.4 0.5.1 – 2019-12-09

- Fixed `setup.py` to package type stubs.

## 3.5 0.5.0 – 2019-12-09

- **BREAKING CHANGE:** `AwsLambda` now requires `Payload` to be a `Parameters` instance.
- `AwsStepFunctions` no longer allows the definition of an execution name.
  - NOTE: This is only *not* a breaking change because `AwsStepFunctions` was fundamentally broken before.
- **BREAKING CHANGE:** `AwsBatch` now required `Parameters` to be a `Parameters` instance.
- **BREAKING CHANGE:** All parameters for `State` classes other than `title` are now keyword-only. #47
- **BREAKING CHANGE:** Most parameters are now keyword-only. #47
- Added explicit local defaults for common State fields: #50
  - `InputPath`
  - `OutputPath`
  - `ResultPath`

### 3.5.1 features

- Add `State.promote` method for states that support `ResultPath`. #32

### 3.5.2 bugfixes

- Fixed `AwsStepFunctions` parameters names. #45

## 3.6 0.4.0 – 2019-12-03

- **BREAKING CHANGE:** Renamed `State.name` to `State.title` #38

## 3.7 0.3.1 – 2019-11-24

- Add support for "comparing" VariablePath instances to Enum members #29
- Add support for troposphere objects as resource values #33
- Initial implementation of context object helper #34

## 3.8 0.3.0 – 2019-11-19

- Add preliminary service integration helpers #27

## 3.9 0.2.1 – 2019-11-18

- Updated docs and added examples.

## 3.10 0.2.0 – 2019-11-17

- **BREAKING CHANGE:** Renamed `ChoiceRule.then_` to `ChoiceRule.then` #12
- **BREAKING CHANGE:** Reworked `Variable` into `JsonPath` and `VariablePath` #3 #10 #11

## 3.11 0.1.0 – 2019-11-14

Initial alpha release.

Includes:

- Basic state/machine construction
- Improved ergonomics for state machines.
- Improved ergonomics for Choice, Task, and Parallel states.

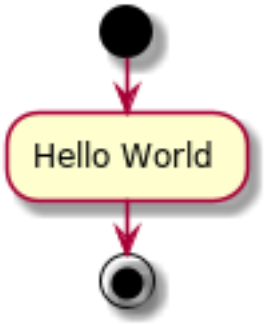
# CHAPTER 4

# rhodes by example

## 4.1 Hello World

This example shows a very simple state machine, with a single state that invokes a Lambda Function Task state.

The `troposphere` version demonstrates how Troposphere objects can be passed as references rather than having to provide a string value.




basic

python

```
"""
A simple hello-world workflow with a single Lambda Task state.
"""
from rhodes.states import StateMachine, Task


def build() -> StateMachine:
    workflow = StateMachine(Comment="A simple minimal example of the States language")

    workflow.start_with(
```

(continues on next page)

(continued from previous page)

```
        Task(
            "Hello World",
            Resource="arn:aws:lambda:us-east-1:123456789012:function:HelloWorld",
        )
    ).end()

    return workflow
```

json

```
{
    "Comment": "A simple minimal example of the States language",
    "StartAt": "Hello World",
    "States": {
        "Hello World": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloWorld",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        }
    }
}
```

troposphere

python

```
"""
A simple hello-world workflow with a single Lambda Task state.
"""
from troposphere import awslambda

from rhodes.states import StateMachine, Task


def build() -> StateMachine:
    lambda_function = awslambda.Function(
        "HelloWorldFunction", Code=awslambda.Code(ZipFile="foo bar")
    )

    workflow = StateMachine(Comment="A simple minimal example of the States language")

    workflow.start_with(Task("Hello World", Resource=lambda_function)).end()

    return workflow
```

json

```
{
    "Comment": "A simple minimal example of the States language",
    "StartAt": "Hello World",
    "States": {
        "Hello World": {
            "Type": "Task",
            "Resource": "${HelloWorldFunction.Arn}",
```

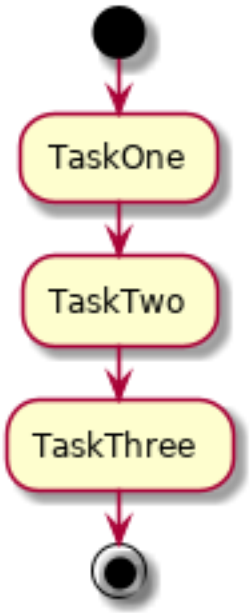
(continues on next page)

(continued from previous page)

```
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        }
    }
}
```

## 4.2 Three Tasks

This example demonstrates a simple state machine composes of three serially executed Task states with no branching.




python

```
"""
Simple workflow with three sequential Lambda Function tasks.
"""
from rhodes.states import StateMachine, Task


def build() -> StateMachine:
    workflow = StateMachine(Comment="This is a state machine with three simple tasks.
↪")

    workflow.start_with(
        Task("TaskOne", Resource="arn:aws:lambda:us-east-1:123456789012:function:One")
    ).then(
        Task("TaskTwo", Resource="arn:aws:lambda:us-east-1:123456789012:function:Two")
    ).then(
        Task(
            "TaskThree", Resource="arn:aws:lambda:us-east-
↪1:123456789012:function:Three"
```

(continues on next page)

(continued from previous page)

```
        )
    ).end()

    return workflow
```

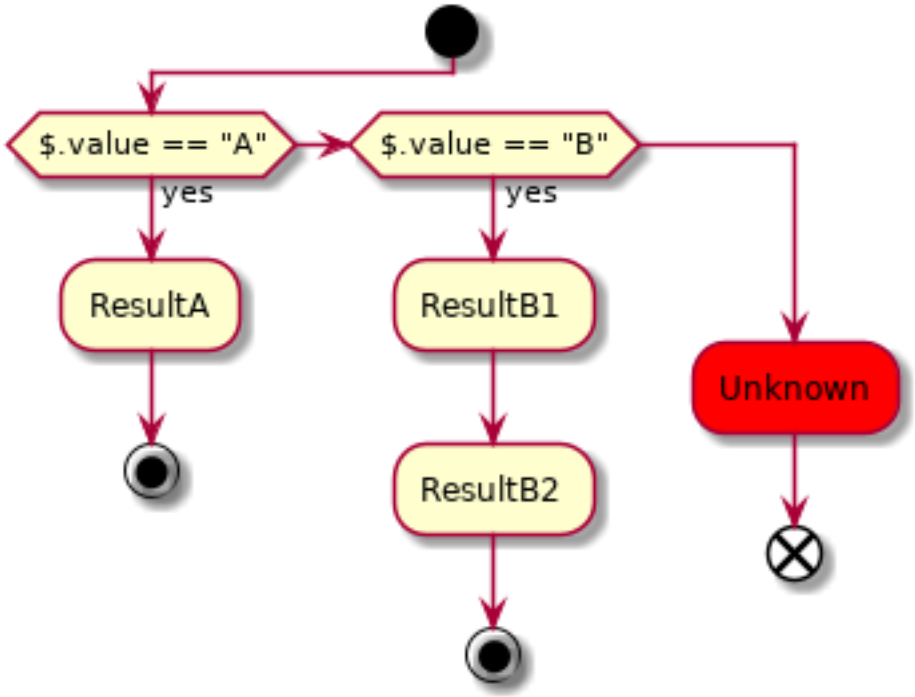
json

```
{
    "Comment": "This is a state machine with three simple tasks.",
    "StartAt": "TaskOne",
    "States": {
        "TaskOne": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:One",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "Next": "TaskTwo"
        },
        "TaskTwo": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Two",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "Next": "TaskThree"
        },
        "TaskThree": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Three",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        }
    }
}
```

## 4.3 Simple Choice

This example shows a simple state machine containing a single Choice state that maps to a small number of terminal states.

The `enums` version demonstrates how Enum members can be used rather than just primitive values.




basic

python

```
"""
Simple workflow using a choice with three possible branches.
"""
from rhodes.choice_rules import VariablePath
from rhodes.states import Choice, Fail, StateMachine, Task


def build() -> StateMachine:
    workflow = StateMachine(
        Comment="This is a simple state machine with a single choice and three end␣
↪states."
    )

    decision = workflow.start_with(Choice("TheBeginning"))

    decision.if_(VariablePath("$.value") == "A").then(
        Task("ResultA", Resource="arn:aws:lambda:us-east-1:123456789012:function:A")
    ).end()

    decision.if_(VariablePath("$.value") == "B").then(
        Task("ResultB1", Resource="arn:aws:lambda:us-east-1:123456789012:function:B1")
    ).then(
        Task("ResultB2", Resource="arn:aws:lambda:us-east-1:123456789012:function:B2")
    ).end()

    decision.else_(Fail("Unknown", Error="Unhandled Case", Cause="Unknown Value"))

    return workflow
```

json

```
{
    "Comment": "This is a simple state machine with a single choice and three end
↪states.",
    "StartAt": "TheBeginning",
    "States": {
        "TheBeginning": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.value",
                    "StringEquals": "A",
                    "Next": "ResultA"
                },
                {
                    "Variable": "$.value",
                    "StringEquals": "B",
                    "Next": "ResultB1"
                }
            ],
            "InputPath": "$",
            "OutputPath": "$",
            "Default": "Unknown"
        },
        "ResultA": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:A",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        },
        "ResultB1": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:B1",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "Next": "ResultB2"
        },
        "ResultB2": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:B2",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        },
        "Unknown": {
            "Type": "Fail",
            "Error": "Unhandled Case",
            "Cause": "Unknown Value"
        }
    }
}
```

enums

python

```python
"""
Simple workflow using a choice with three possible branches.
"""
from enum import Enum

from rhodes.choice_rules import VariablePath
from rhodes.states import Choice, Fail, StateMachine, Task


class Values(Enum):
    CHOICE_A = "A"
    CHOICE_B = "B"


def build() -> StateMachine:
    workflow = StateMachine(
        Comment="This is a simple state machine with a single choice and three end␣
↪states."
    )

    decision = workflow.start_with(Choice("TheBeginning"))
    decision.if_(VariablePath("$.value") == Values.CHOICE_A).then(
        Task(
            "ResultA", Resource="arn:aws:lambda:us-east-1:123456789012:function:A"
        ).end()
    )
    result_b1 = decision.if_(VariablePath("$.value") == Values.CHOICE_B).then(
        Task("ResultB1", Resource="arn:aws:lambda:us-east-1:123456789012:function:B1")
    )
    result_b1.then(
        Task(
            "ResultB2", Resource="arn:aws:lambda:us-east-1:123456789012:function:B2"
        ).end()
    )
    decision.else_(Fail("Unknown", Error="Unhandled Case", Cause="Unknown Value"))

    return workflow
```

json

```json
{
    "Comment": "This is a simple state machine with a single choice and three end␣
↪states.",
    "StartAt": "TheBeginning",
    "States": {
        "TheBeginning": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.value",
                    "StringEquals": "A",
                    "Next": "ResultA"
                },
                {
                    "Variable": "$.value",
                    "StringEquals": "B",
```

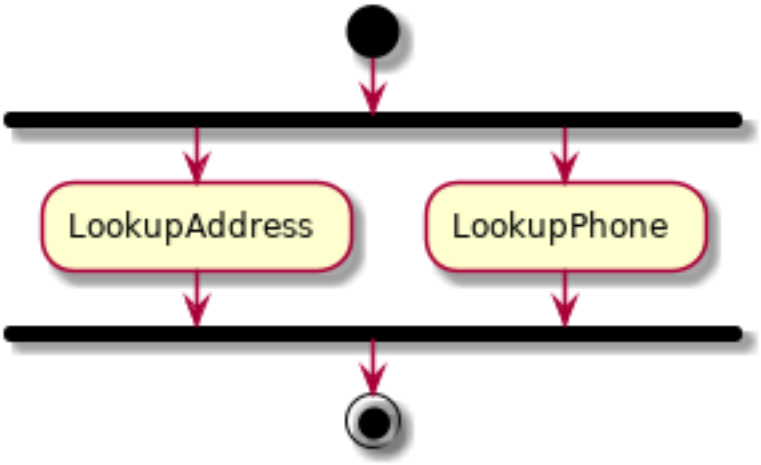
(continues on next page)

(continued from previous page)

```
                    "Next": "ResultB1"
                }
            ],
            "InputPath": "$",
            "OutputPath": "$",
            "Default": "Unknown"
        },
        "ResultA": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:A",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        },
        "ResultB1": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:B1",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "Next": "ResultB2"
        },
        "ResultB2": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:B2",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true
        },
        "Unknown": {
            "Type": "Fail",
            "Error": "Unhandled Case",
            "Cause": "Unknown Value"
        }
    }
}
```

## 4.4 Simple Parallel

This example demonstrates creating a simple state machine containing a single Parallel state that runs some Task states.




python

```
"""
Simple workflow using a Parallel state with two concurrent workflows.
"""
from rhodes.states import Parallel, StateMachine, Task


def build() -> StateMachine:
    lookup_address = StateMachine()
    lookup_address.start_with(
        Task(
            "LookupAddress",
            Resource="arn:aws:lambda:us-east-1:123456789012:function:AddressFinder",
        )
    ).end()

    lookup_phone = StateMachine()
    lookup_phone.start_with(
        Task(
            "LookupPhone",
            Resource="arn:aws:lambda:us-east-1:123456789012:function:PhoneFinder",
        )
    ).end()

    parallel_run = Parallel("LookupCustomerInfo")
    parallel_run.add_branch(lookup_address)
    parallel_run.add_branch(lookup_phone)

    workflow = StateMachine(Comment="Parallel Example.")
    workflow.start_with(parallel_run).end()

    return workflow
```

json

```
{
    "Comment": "Parallel Example.",
    "StartAt": "LookupCustomerInfo",
    "States": {
```

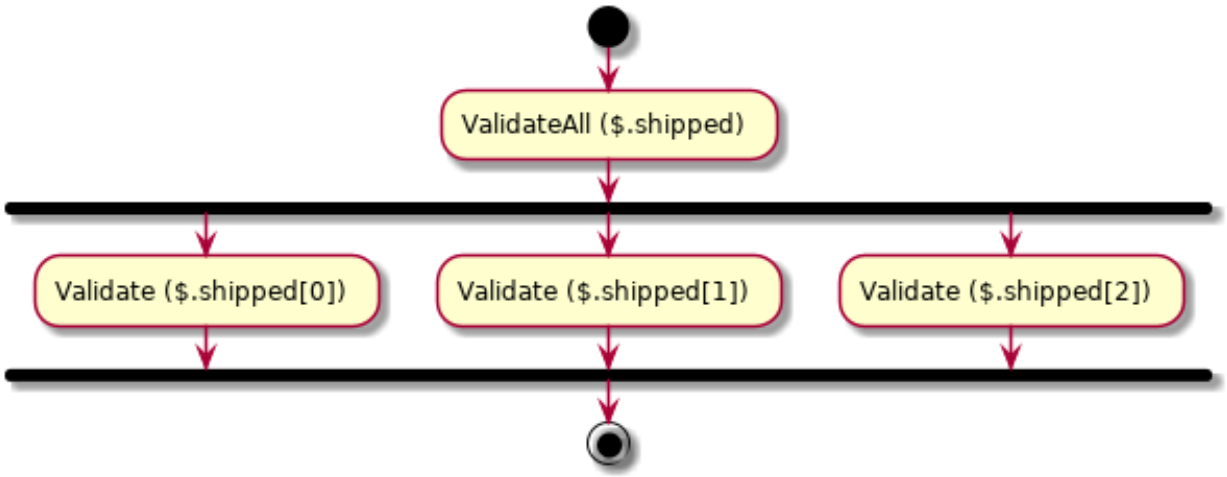
(continues on next page)

(continued from previous page)

```
        "LookupCustomerInfo": {
            "Type": "Parallel",
            "InputPath": "$",
            "OutputPath": "$",
            "ResultPath": "$",
            "End": true,
            "Branches": [
                {
                    "StartAt": "LookupAddress",
                    "States": {
                        "LookupAddress": {
                            "Type": "Task",
                            "Resource": "arn:aws:lambda:us-east-
↪1:123456789012:function:AddressFinder",
                            "InputPath": "$",
                            "OutputPath": "$",
                            "ResultPath": "$",
                            "End": true
                        }
                    }
                },
                {
                    "StartAt": "LookupPhone",
                    "States": {
                        "LookupPhone": {
                            "Type": "Task",
                            "Resource": "arn:aws:lambda:us-east-
↪1:123456789012:function:PhoneFinder",
                            "InputPath": "$",
                            "OutputPath": "$",
                            "ResultPath": "$",
                            "End": true
                        }
                    }
                }
            ]
        }
    }
}
```

## 4.5 Simple Map

This example demonstrates using the Map state to run a Task over every member of a part of the state machine data.




python

```python
"""
Simple workflow using the Map state.
"""
from rhodes.states import Map, StateMachine, Task
from rhodes.structures import ContextPath, JsonPath, Parameters


def build() -> StateMachine:
    validate_task = Task(
        "Validate", Resource="arn:aws:lambda:us-east-1:123456789012:function:ship-val"
    )

    state_iterator = StateMachine()
    state_iterator.start_with(validate_task).end()

    mapper = Map(
        "ValidateAll",
        InputPath=JsonPath("$.detail"),
        ItemsPath=JsonPath("$.shipped"),
        Parameters=Parameters(
            Execution=ContextPath().Execution.Id, Payload=JsonPath("$")
        ),
        MaxConcurrency=0,
        Iterator=state_iterator,
        ResultPath=JsonPath("$.detail.shipped"),
    )

    workflow = StateMachine(Comment="Simple state machine with one map state")
    workflow.start_with(mapper).end()

    return workflow
```

json

```
{
    "Comment": "Simple state machine with one map state",
    "StartAt": "ValidateAll",
    "States": {
```

(continues on next page)

(continued from previous page)

```
        "ValidateAll": {
            "Type": "Map",
            "InputPath": "$.detail",
            "ItemsPath": "$.shipped",
            "MaxConcurrency": 0,
            "Parameters": {
                "Execution.$": "$$.Execution.Id",
                "Payload.$": "$"
            },
            "Iterator": {
                "StartAt": "Validate",
                "States": {
                    "Validate": {
                        "Type": "Task",
                        "Resource": "arn:aws:lambda:us-east-
↪1:123456789012:function:ship-val",
                        "InputPath": "$",
                        "OutputPath": "$",
                        "ResultPath": "$",
                        "End": true
                    }
                }
            },
            "ResultPath": "$.detail.shipped",
            "OutputPath": "$",
            "End": true
            }
    }
}
```

# CHAPTER 5

# API Reference

## 5.1 states

### 5.1.1 Step Functions States

Standard Step Functions and state machine states.

See Step Functions docs for more details.

**class** rhodes.states.**State**(*title*, *, *Comment=None*)

Bases: object

Base class for states.

**member_of = None**

**to_dict**()

Serialize state as a dictionary.

**Return type** Dict

**promote**(*path*)

Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**class** rhodes.states.**StateMachine**(*, *States=NOTHING*, *StartAt=None*, *Comment=None*, *Version=None*, *TimeoutSeconds=None*)

Bases: object

Step Functions State Machine.

See Step Functions docs for more details.

**Parameters**

- **States** (*dict(str, State)*) – Map of states that make up this state machine
- **StartAt** (*str*) – The state where this state machine starts
- **Comment** (*str*) – Human-readable description of the state
- **Version** (*str*) – The version of the Amazon States Language used in this state machine (must be `1.0` if provided)
- **TimeoutSeconds** (*int*) – Maximum time that this state machine is allowed to run

**to_dict**()
Serialize this state machine as a dictionary.

**Return type** Dict

**definition_string**()
Serialize this state machine for use in a `troposphere` state machine definition.

**Return type** Sub

**add_state**(*new_state*)
Add a state to this state machine.

**Parameters** **new_state** (State) – State to add

**Return type** *State*

**start_with**(*first_state*)
Add a state to this state machine and mark it as the starting state.

**Parameters** **first_state** (State) – State to start with

**Return type** *State*

**class** rhodes.states.**Pass**(*title*, *, *Comment=None*, *Result=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Parameters=None*)
Bases: *rhodes.states.State*

A Pass state passes its input to its output without performing work. Pass states are useful when constructing and debugging state machines.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Parameters** (Parameters) – Additional parameters for Step Functions to provide to connected resource

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.**Task**(*title*, *, *Comment=None*, *Resource=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Parameters=None*)
Bases: *rhodes.states.State*

A Task state represents a single unit of work performed by a state machine.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Parameters** (Parameters) – Additional parameters for Step Functions to provide to connected resource

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.**Choice**(*title*, *, *Comment=None*, *Choices=NOTHING*, *Default=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*)
Bases: *rhodes.states.State*

A Choice state adds branching logic to a state machine.

See Step Functions docs for more details.

```
workflow = StateMachine()

next_state = Pass("Ok number")

decision = workflow.start_with(Choice("Make a decision"))
decision.if_(VariablePath("$.foo.bar") == 12).then(next_state)
decision.if_(VariablePath("$.foo.bar") < 0).then(Fail("Negative value!"))
decision.if_(all_(
    VariablePath("$.foo.bar") != 12,
    VariablePath("$.baz.wow") == "not 12!",
)).then(next_state)
decision.else_(Succeed("Something else!"))

next_state.end()
```

```
{
    "States": {
        "Make a decision": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.foo.bar",
                    "NumericEquals": 12,
                    "Next": "Ok number"
                },
                {
                    "Variable": "$.foo.bar",
                    "NumericLessThan": 0,
                    "Next": "Negative value!"
                },
                {
                    "And": [
                        {
                            "Not": {
                                "Variable": "$.foo.bar",
                                "NumericEquals": 12
```

(continues on next page)

(continued from previous page)

```
                            }
                        },
                        {
                            "Variable": "$.baz.wow",
                            "StringEquals": "not 12!"
                        }
                    ],
                    "Next": "Ok number"
                }
            ],
            "Default": "Something else!"
        },
        "Ok number": {
            "Type": "Pass",
            "End": true
        },
        "Negative value!": {
            "Type": "Fail"
        },
        "Something else!": {
            "Type": "Succeed"
        }
    }
}
```

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))

**add_choice**(*rule*)

Add a choice rule to this state. This is the lower-level interface that **:method:'if_'** uses.

**Parameters** **rule** (*ChoiceRule*) – Rule to add

**Return type** *ChoiceRule*

**Returns** rule

**if_**(*rule*)

Add a choice rule to this state as one possible logic branch. This should be followed up with a .then(STATE) call.

```
decision.if_(VariablePath("$.foo.bar") == 12).then(next_state)
```

This results in the rule definition:

```
{
    "Variable": "$.foo.bar",
    "NumericEquals": 12,
    "Next": "NEXT_STATE_NAME"
}
```

**Parameters** **rule** (*ChoiceRule*) – The rule to add

**Return type** *ChoiceRule*

**Returns** rule

**else_**(*state*)
Add a default state. This is the state to transition to if none of the choice rules are satisfied.

**Parameters** **state** (*State*) – The default state to add

**Return type** *State*

**Returns** state

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**class** rhodes.states.**Wait**(*title*, *, *Comment=None*, *Seconds=None*, *Timestamp=None*, *SecondsPath=None*, *TimestampPath=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*)
Bases: *rhodes.states.State*

A Wait state delays the state machine from continuing for a specified time. You can choose either a relative time, specified in seconds from when the state begins, or an absolute end time, specified as a timestamp.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.**Succeed**(*title*, *, *Comment=None*)
Bases: *rhodes.states.State*

A Succeed state stops an execution successfully. The Succeed state is a useful target for Choice state branches that don't do anything but stop the execution.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.**Fail**(*title*, *, *Comment=None*, *Error=None*, *Cause=None*)
Bases: *rhodes.states.State*

A Fail state stops the execution of the state machine and marks it as a failure.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.**Parallel**(*title*, ***, *Comment=None*, *Branches=NOTHING*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *Parameters=None*)

Bases: *rhodes.states.State*

The Parallel state can be used to create parallel branches of execution in your state machine.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **Parameters** (Parameters) – Additional parameters for Step Functions to provide to connected resource

**to_dict**()

Serialize state as a dictionary.

**Return type** Dict

**add_branch**(*state_machine=None*)

Add a parallel branch to this state. If state_machine is not provided, we generate an empty state machine and add that.

**Parameters** **state_machine** (Optional[*StateMachine*]) – State machine to add (optional)

**Return type** *StateMachine*

**Returns** state_machine if provided or a new empty state machine if not

**end**()

Make this state a terminal state.

**member_of = None**

**promote**(*path*)

Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**class** rhodes.states.**Map**(*title*, ***, *Comment=None*, *Iterator=None*, *ItemsPath=None*, *MaxConcurrency=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *Parameters=None*)
Bases: *rhodes.states.State*

The Map state can be used to run a set of steps for each element of an input array. While the Parallel state executes multiple branches of steps using the same input, a Map state will execute the same steps for multiple entries of an array in the state input.

See Step Functions docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **Parameters** (Parameters) – Additional parameters for Step Functions to provide to connected resource

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a *Pass* state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**()
Serialize state as a dictionary.

**Return type** Dict

### 5.1.2 Service Integration Helpers

AWS Service Integrations for Task states.

https://docs.aws.amazon.com/step-functions/latest/dg/concepts-service-integrations.html

All fields that would normally be collected in the `Parameters` parameter are collapsed to the main instantiation level for all `ServiceIntegration` classes. They are internally collected into a `Parameters` instance that is assigned to the `Parameters` parameter when these are converted into `Task` instances.

The rules for these values are as follows:

- If Step Functions expects a single value, the value can be a string, a `JsonPath`, or a `troposphere.AWSHelperFn`.
- If Step Functions can accept a complex value, the value can be a `JsonPath`, `Parameters`, or a `troposphere.AWSHelperFn`.
- If the value represents an AWS resource, such as a Lambda Function or Step Functions State Machine, the value can be the appropriate Troposphere type.

Any of these values can also be an `Enum`. If you choose to use an `Enum` and the instance value is NOT one of the above appropriate types, the serialization will fail.

#### AWS Lambda

AWS Lambda Task state.

**class** rhodes.states.services.awslambda.**AwsLambda**(*title*, *, *Comment=None*, *FunctionName=None*, *Payload=None*, *ClientContext=None*, *InvocationType=None*, *Qualifier=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ">*)

Bases: *rhodes.states.State*

Invoke a Lambda function.

See service docs for more details.

**Parameters**

- **FunctionName** – AWS Lambda Function to call
- **Payload** (`Parameters`, `JsonPath`, `AWSHelperFn`, dict, str, or `Enum`) – Data to provide to the Lambda Function as input
- **ClientContext** (`JsonPath`, `AWSHelperFn`, str, or `Enum`) – Up to 3583 bytes of base64-encoded data about the invoking client to pass to the function in the context object
- **InvocationType** (`JsonPath`, `AWSHelperFn`, str, or `Enum`) – Determines how the Lambda Function is invoked

- **Qualifier** (JsonPath, AWSHelperFn, str, or Enum) – Version or alias of the Lambda Function to invoke
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### AWS Batch

AWS Batch Task state.

**class** rhodes.states.services.batch.**AwsBatch**(*title*, ***, *Comment=None*, *JobDefinition=None*, *JobName=None*, *JobQueue=None*, *Parameters=None*, *ArrayProperties=None*, *ContainerOverrides=None*, *DependsOn=None*, *RetryStrategy=None*, *Timeout=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ”>*)

Bases: *rhodes.states.State*

Submit an AWS Batch job from a job definition.

See service docs for more details.

**Parameters**

- **JobDefinition** – The job definition used by this job. This value can be one of name, name:revision, or the Amazon Resource Name (ARN) for the job definition. If name is specified without a revision then the latest active revision is used.
- **JobName** – The name of the job. The first character must be alphanumeric, and up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.
- **JobQueue** – The job queue into which the job is submitted. You can specify either the name or the Amazon Resource Name (ARN) of the queue.
- **Parameters** – Additional parameters passed to the job. These replace parameter substitution placeholders that are set in the job definition. Parameters are specified as a key and value pair mapping. Parameters in a SubmitJob request override any corresponding parameter defaults from the job definition.
- **ArrayProperties** – The array properties for the submitted job, such as the size of the array. The array size can be between 2 and 10,000. If you specify array properties for a job, it becomes an array job.
- **ContainerOverrides** – A list of container overrides in JSON format. These specify the name of a container in the specified job definition and the overrides it should receive. You can override the default command for a container with a command override. You can also override existing environment variables on a container or add new environment variables to it with an environment override.
- **DependsOn** – A list of dependencies for the job. A job can depend upon a maximum of 20 jobs.
- **RetryStrategy** – The retry strategy to use for failed jobs from this SubmitJob operation. When a retry strategy is specified here, it overrides the retry strategy defined in the job definition.
- **Timeout** – The timeout configuration for this SubmitJob operation. If a job is terminated due to a timeout, it is not retried. The minimum value for the timeout is 60 seconds. This configuration overrides any timeout configuration specified in the job definition. For array jobs, child jobs have the same timeout configuration as the parent job.
- **title** (*str*) – Name of state in state machine

- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### Amazon DynamoDB

Amazon DynamoDB Task states.

**class** rhodes.states.services.dynamodb.**AmazonDynamoDb**(*, *TableName=None*)
Bases: object

Helper to provide easy access to integration helpers.

**Parameters** **TableName** – The table to interact with

**get_item**(*title*, ***kwargs*)
Return a set of attributes for the item with the given primary key.

TableName as well as any provided kwargs are passed to the *AmazonDynamoDbGetItem* constructor.

**Return type** *AmazonDynamoDbGetItem*

**put_item**(*title*, ***kwargs*)
Create a new item or replace an old item with a new item.

`TableName` as well as any provided kwargs are passed to the *AmazonDynamoDbPutItem* constructor.

**Return type** *AmazonDynamoDbPutItem*

**delete_item**(*title*, ***kwargs*)
Delete a single item in a table by primary key.

`TableName` as well as any provided kwargs are passed to the *AmazonDynamoDbDeleteItem* constructor.

**Return type** *AmazonDynamoDbDeleteItem*

**update_item**(*title*, ***kwargs*)
Edit an existing item's attributes or add a new item to the table if it does not already exist.

`TableName` as well as any provided kwargs are passed to the *AmazonDynamoDbUpdateItem* constructor.

**Return type** *AmazonDynamoDbUpdateItem*

**class** rhodes.states.services.dynamodb.**AmazonDynamoDbGetItem**(*title*, *, *Comment=None*, *AttributesToGet=None*, *ConsistentRead=None*, *ExpressionAttributeNames=None*, *ProjectionExpression=None*, *ReturnConsumedCapacity=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPO ">*, *TableName=None*, *Key=None*)

Bases: *rhodes.states.State*

Return a set of attributes for the item with the given primary key.

See service docs for more details.

**Parameters**

- **AttributesToGet** – This is a legacy parameter. Use ProjectionExpression instead. For more information, see AttributesToGet in the Amazon DynamoDB Developer Guide.

- **ConsistentRead** – Determines the read consistency model
- **ExpressionAttributeNames** – One or more substitution tokens for attribute names in an expression.
- **ProjectionExpression** – A string that identifies one or more attributes to retrieve from the table.
- **ReturnConsumedCapacity** – Determines the level of detail about provisioned throughput consumption that is returned in the response
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **TableName** – The table to interact with
- **Key** – A map of attribute names to AttributeValue objects, representing the primary key of the item to retrieve.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.dynamodb.**AmazonDynamoDbPutItem**(*title*, ***, *Comment=None*, *Item=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPO ">*, *TableName=None*, *ConditionalOperator=None*, *ConditionExpression=None*, *Expected=None*, *ExpressionAttributeNames=None*, *ExpressionAttributeValues=None*, *ReturnConsumedCapacity=None*, *ReturnItemCollectionMetrics=None*, *ReturnValues=None*)

Bases: *rhodes.states.State*

Create a new item or replace an old item with a new item.

See service docs for more details.

**Parameters**

- **Item** – A map of attribute name/value pairs, one for each attribute.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –

- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **TableName** – The table to interact with
- **ConditionalOperator** – This is a legacy parameter. Use ConditionExpression instead. For more information, see ConditionalOperator in the Amazon DynamoDB Developer Guide.
- **ConditionExpression** – A condition that must be satisfied in order for a conditional operation to succeed.
- **Expected** – This is a legacy parameter. Use ConditionExpression instead. For more information, see Expected in the Amazon DynamoDB Developer Guide.
- **ExpressionAttributeNames** – One or more substitution tokens for attribute names in an expression.
- **ExpressionAttributeValues** – One or more values that can be substituted in an expression.
- **ReturnConsumedCapacity** – Determines the level of detail about provisioned throughput consumption that is returned in the response
- **ReturnItemCollectionMetrics** – Determines whether item collection metrics are returned.
- **ReturnValues** – Use ReturnValues if you want to get the item attributes as they appeared before they were updated with the request.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.dynamodb.**AmazonDynamoDbDeleteItem**(*title*, ***, *Comment=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RE ">*, *TableName=None*, *Key=None*, *ConditionalOperator=None*, *ConditionExpression=None*, *Expected=None*, *ExpressionAttributeNames=None*, *ExpressionAttributeValues=None*, *ReturnConsumedCapacity=None*, *ReturnItemCollectionMetrics=None*, *ReturnValues=None*)

Bases: *rhodes.states.State*

Delete a single item in a table by primary key.

See service docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))

- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **TableName** – The table to interact with
- **Key** – A map of attribute names to AttributeValue objects, representing the primary key of the item to retrieve.
- **ConditionalOperator** – This is a legacy parameter. Use ConditionExpression instead. For more information, see ConditionalOperator in the Amazon DynamoDB Developer Guide.
- **ConditionExpression** – A condition that must be satisfied in order for a conditional operation to succeed.
- **Expected** – This is a legacy parameter. Use ConditionExpression instead. For more information, see Expected in the Amazon DynamoDB Developer Guide.
- **ExpressionAttributeNames** – One or more substitution tokens for attribute names in an expression.
- **ExpressionAttributeValues** – One or more values that can be substituted in an expression.
- **ReturnConsumedCapacity** – Determines the level of detail about provisioned throughput consumption that is returned in the response
- **ReturnItemCollectionMetrics** – Determines whether item collection metrics are returned.
- **ReturnValues** – Use ReturnValues if you want to get the item attributes as they appeared before they were updated with the request.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.dynamodb.**AmazonDynamoDbUpdateItem**(*title*, *, *Comment=None*, *AttributeUpdates=None*, *UpdateExpression=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RE ">*, *TableName=None*, *Key=None*, *ConditionalOperator=None*, *ConditionExpression=None*, *Expected=None*, *ExpressionAttributeNames=None*, *ExpressionAttributeValues=None*, *ReturnConsumedCapacity=None*, *ReturnItemCollectionMetrics=None*, *ReturnValues=None*)

Bases: *rhodes.states.State*

Edit an existing item's attributes or add a new item to the table if it does not already exist.

See service docs for more details.

**Parameters**

- **AttributeUpdates** – This is a legacy parameter. Use UpdateExpression instead. For more information, see AttributeUpdates in the Amazon DynamoDB Developer Guide.
- **UpdateExpression** – An expression that defines one or more attributes to be updated, the action to be performed on them, and new values for them.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)
- **TableName** – The table to interact with
- **Key** – A map of attribute names to AttributeValue objects, representing the primary key of the item to retrieve.
- **ConditionalOperator** – This is a legacy parameter. Use ConditionExpression instead. For more information, see ConditionalOperator in the Amazon DynamoDB Developer Guide.
- **ConditionExpression** – A condition that must be satisfied in order for a conditional operation to succeed.
- **Expected** – This is a legacy parameter. Use ConditionExpression instead. For more information, see Expected in the Amazon DynamoDB Developer Guide.
- **ExpressionAttributeNames** – One or more substitution tokens for attribute names in an expression.
- **ExpressionAttributeValues** – One or more values that can be substituted in an expression.
- **ReturnConsumedCapacity** – Determines the level of detail about provisioned throughput consumption that is returned in the response
- **ReturnItemCollectionMetrics** – Determines whether item collection metrics are returned.
- **ReturnValues** – Use ReturnValues if you want to get the item attributes as they appeared before they were updated with the request.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a `Pass` state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (`Union`[`str`, `Enum`, `JSONPath`, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** `Dict`

### Amazon ECS/Fargate

Amazon ECS/Fargate Task state.

**class** rhodes.states.services.ecs.**AmazonEcs**(*title*, *, *Comment=None*, *Cluster=None*, *Group=None*, *LaunchType=None*, *NetworkConfiguration=None*, *Overrides=None*, *PlacementConstraints=None*, *PlacementStrategy=None*, *PlatformVersion=None*, *TaskDefinition=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ”>*)

Bases: *rhodes.states.State*

Start a new task using the specified task definition.

See service docs for more details.

**Parameters**

- **Cluster** – The short name or full Amazon Resource Name (ARN) of the cluster on which to run your task. If you do not specify a cluster, the default cluster is assumed.
- **Group** – The name of the task group to associate with the task. The default value is the family name of the task definition (for example, family:my-family-name).
- **LaunchType** – The launch type on which to run your task. For more information, see Amazon ECS Launch Types in the Amazon Elastic Container Service Developer Guide.
- **NetworkConfiguration** – The network configuration for the task. This parameter is required for task definitions that use the awsvpc network mode to receive their own elastic network interface, and it is not supported for other network modes. For more information, see Task Networking in the Amazon Elastic Container Service Developer Guide.

- **Overrides** – A list of container overrides in JSON format that specify the name of a container in the specified task definition and the overrides it should receive.
- **PlacementConstraints** – An array of placement constraint objects to use for the task. You can specify up to 10 constraints per task (including constraints in the task definition and those specified at runtime).
- **PlacementStrategy** – The placement strategy objects to use for the task. You can specify a maximum of five strategy rules per task.
- **PlatformVersion** – The platform version the task should run. A platform version is only specified for tasks using the Fargate launch type. If one is not specified, the LATEST platform version is used by default. For more information, see AWS Fargate Platform Versions in the Amazon Elastic Container Service Developer Guide.
- **TaskDefinition** – The family and revision (family:revision) or full ARN of the task definition to run. If a revision is not specified, the latest ACTIVE revision is used.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a `Pass` state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### AWS Glue

AWS Glue Task state.

**class** rhodes.states.services.glue.**AwsGlue**(*title*, *, *Comment=None*, *JobName=None*, *JobRunId=None*, *Arguments=None*, *AllocatedCapacity=None*, *Timeout=None*, *SecurityConfiguration=None*, *NotificationProperty=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ">*)

Bases: *rhodes.states.State*

Start a job run using a job definition.

See service docs for more details.

**Parameters**

- **JobName** – The name of the job definition to use.
- **JobRunId** – The ID of a previous JobRun to retry.
- **Arguments** – The job arguments specifically for this run.
- **AllocatedCapacity** – This field is deprecated. Use MaxCapacity instead. The number of AWS Glue data processing units (DPUs) to allocate to this JobRun.
- **Timeout** – The JobRun timeout in minutes.
- **SecurityConfiguration** – The name of the SecurityConfiguration structure to be used with this job run.
- **NotificationProperty** – Specifies configuration properties of a job run notification.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run

- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a `@.` prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### Amazon SageMaker

Amazon SageMaker Task states.

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateEndpoint**(*title*, *, *Comment=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQU ">*, *EndpointConfigName=None*, *EndpointName=None*, *Tags=None*)

Bases: *rhodes.states.State*

Create an endpoint using the endpoint configuration specified in the request.

See service docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –

- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **EndpointConfigName** – The name of an endpoint configuration.
- **EndpointName** – The name of the endpoint.
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateEndpointConfig**(*title*, ***, *Comment=None*, *KmsKeyId=None*, *ProductionVariants=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=I*, *OutputPath=JsonPath(path=I*, *ResultPath=JsonPath(path=I*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPatte">*, *EndpointConfigName=None*, *Tags=None*)

Bases: *rhodes.states.State*

Create an endpoint configuration that Amazon SageMaker hosting services uses to deploy models.

See service docs for more details.

**Parameters**

- **KmsKeyId** – The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance that hosts the endpoint.
- **ProductionVariants** – An list of ProductionVariant objects, one for each model that you want to host at this endpoint.
- **title** (*str*) – Name of state in state machine

- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **EndpointConfigName** – The name of an endpoint configuration.
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateHyperParameterTuningJob**(*title*, ***, *Comment=No HyperParameterTuningJobConfig=None HyperParameterTuningJobName=No TrainingJobDefinition=Non WarmStartConfig=None Next=No End=No InputPath=Jso OutputPath=Jso ResultPath=Jso Retry=No Catch=N TimeoutSeconds=No HeartbeatSeconds=No Pattern=<In ">, Tags=No*

Bases: *rhodes.states.State*

Start a hyperparameter tuning job.

See service docs for more details.

**Parameters**

- **HyperParameterTuningJobConfig** – The HyperParameterTuningJobConfig object that describes the tuning job, including the search strategy, the objective metric used to evaluate training jobs, ranges of parameters to search, and resource limits for the tuning job.
- **HyperParameterTuningJobName** – The name of the tuning job. This name is the prefix for the names of all training jobs that this tuning job launches. The name must be unique within the same AWS account and AWS Region. The name must have { } to { } characters. Valid characters are a-z, A-Z, 0-9, and : + = @ _ % - (hyphen). The name is not case sensitive.
- **TrainingJobDefinition** – The HyperParameterTrainingJobDefinition object that describes the training jobs that this tuning job launches, including static hyperparameters, input data configuration, output data configuration, resource configuration, and stopping condition.
- **WarmStartConfig** – Specifies the configuration for starting the hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()

Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a `Pass` state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (`Union`[`str`, `Enum`, `JSONPath`, *`JsonPath`*]) – Path to promote

**Return type** *`Pass`*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** `Dict`

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateLabelingJob**(*title*, ***, *Comment=None*, *HumanTaskConfig=None*, *InputConfig=None*, *LabelAttributeName=None*, *LabelCategoryConfigS3Uri=None*, *LabelingJobAlgorithmsConfig=None*, *LabelingJobName=None*, *OutputConfig=None*, *RoleArn=None*, *StoppingConditions=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root(*
*OutputPath=JsonPath(path=Root(*
*ResultPath=JsonPath(path=Root(*
*Retry=None*, *Catch=None*, *TimeoutSec-*

Bases: *rhodes.states.State*

Create a job that uses workers to label the data objects in your input dataset.

See service docs for more details.

**Parameters**

- **HumanTaskConfig** – Configures the labeling task and how it is presented to workers; including, but not limited to price, keywords, and batch size (task count).
- **InputConfig** – Input data for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.
- **LabelAttributeName** – The attribute name to use for the label in the output manifest file.
- **LabelCategoryConfigS3Uri** – The S3 URL of the file that defines the categories used to label the data objects.
- **LabelingJobAlgorithmsConfig** – Configures the information required to perform automated data labeling.
- **LabelingJobName** – The name of the labeling job. This name is used to identify the job in a list of labeling jobs.
- **OutputConfig** – The location of the output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.
- **RoleArn** – The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete data labeling.
- **StoppingConditions** – A set of conditions for stopping the labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)

- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateModel**(*title*, *, *Comment=None*, *Containers=None*, *EnableNetworkIsolation=None*, *ExecutionRoleArn=None*, *ModelName=None*, *PrimaryContainer=None*, *VpcConfig=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST"*>, *Tags=None*)

Bases: *rhodes.states.State*

Create a model in Amazon SageMaker.

See service docs for more details.

**Parameters**

- **Containers** – Specifies the containers in the inference pipeline.
- **EnableNetworkIsolation** – Isolates the model container. No inbound or outbound network calls can be made to or from the model container.
- **ExecutionRoleArn** – The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute instances or for batch transform jobs. Deploying on ML compute instances is part of model hosting. For more information, see Amazon SageMaker Roles.
- **ModelName** – The name of the new model.
- **PrimaryContainer** – The location of the primary docker image containing inference

code, associated artifacts, and custom environment map that the inference code uses when the model is deployed for predictions.

- **VpcConfig** – A VpcConfig object that specifies the VPC that you want your model to connect to.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateTrainingJob**(*title*, ***, *Comment=None*, *AlgorithmSpecification=None*, *HyperParameters=None*, *InputDataConfig=None*, *OutputDataConfig=None*, *ResourceConfig=None*, *RoleArn=None*, *StoppingCondition=None*, *TrainingJobName=None*, *VpcConfig=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root(*
*OutputPath=JsonPath(path=Root(*
*ResultPath=JsonPath(path=Root(*
*Retry=None*, *Catch=None*, *TimeoutSec-*

Bases: *rhodes.states.State*

Start a model training job.

See service docs for more details.

**Parameters**

- **AlgorithmSpecification** – The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode.
- **HyperParameters** – Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process.
- **InputDataConfig** – An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.
- **OutputDataConfig** – Specifies the path to the S3 location where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.
- **ResourceConfig** – The resources, including the ML compute instances and ML storage volumes, to use for model training.
- **RoleArn** – The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.
- **StoppingCondition** – Specifies a limit to how long a model training job can run. When the job reaches the time limit, Amazon SageMaker ends the training job. Use this API to cap model training costs.
- **TrainingJobName** – The name of the training job. The name must be unique within an AWS Region in an AWS account.
- **VpcConfig** – A VpcConfig object that specifies the VPC that you want your training job to connect to.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)

- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**class** rhodes.states.services.sagemaker.**AmazonSageMakerCreateTransformJob**(*title*, ***, *Comment=None*, *BatchStrategy=None*, *Environment=None*, *MaxConcurrentTransforms=None*, *MaxPayloadInMB=None*, *ModelName=None*, *TransformInput=None*, *TransformJobName=None*, *TransformOutput=None*, *TransformResources=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Roo*, *OutputPath=JsonPath(path=Roo*, *ResultPath=JsonPath(path=Roo*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*,

Bases: *rhodes.states.State*

Start a transform job.

See service docs for more details.

**Parameters**

- **BatchStrategy** – Specifies the number of records to include in a mini-batch for an HTTP inference request. A record is a single unit of input data that inference can be made on. For example, a single line in a CSV file is a record.
- **Environment** – The environment variables to set in the Docker container.
- **MaxConcurrentTransforms** – The maximum number of parallel requests that can be sent to each instance in a transform job.
- **MaxPayloadInMB** – The maximum allowed size of the payload, in MB.
- **ModelName** – The name of the model that you want to use for the transform job. ModelName must be the name of an existing Amazon SageMaker model within an AWS Region in an AWS account.
- **TransformInput** – Describes the input source and the way the transform job consumes it.
- **TransformJobName** – The name of the transform job. The name must be unique within an AWS Region in an AWS account.
- **TransformOutput** – Describes the results of the transform job.
- **TransformResources** – Describes the resources, including ML instance types and ML instance count, to use for the transform job.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: `<IntegrationPattern.REQUEST_RESPONSE: ''>`)
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a `Pass` state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (`Union`[`str`, `Enum`, `JSONPath`, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** `Dict`

**class** rhodes.states.services.sagemaker.**AmazonSageMakerUpdateEndpoint**(*title*, *, *Comment=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQU ">*, *EndpointConfigName=None*, *EndpointName=None*, *Tags=None*)

Bases: *rhodes.states.State*

Deploy the new EndpointConfig specified in the request. Then switch to using newly created endpoint and delete resources provisioned for the endpoint using the previous EndpointConfig.

See service docs for more details.

**Parameters**

- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)
- **EndpointConfigName** – The name of an endpoint configuration.
- **EndpointName** – The name of the endpoint.
- **Tags** – An array of key-value pairs. For more information, see Using Cost Allocation Tagsin the AWS Billing and Cost Management User Guide.

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### Amazon SNS

Amazon SNS Task state.

**class** rhodes.states.services.sns.**AmazonSns**(*title*, ***, *Comment=None*, *Message=None*, *MessageAttributes=None*, *MessageStructure=None*, *Subject=None*, *PhoneNumber=None*, *TargetArn=None*, *TopicArn=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ">*)

Bases: *rhodes.states.State*

Send a message to target using Amazon SNS. Target can be an Amazon SNS topic, a text message (SMS message) directly to a phone number, or a message to a mobile platform endpoint (when you specify the TargetArn).

See service docs for more details.

Exactly one of `PhoneNumber`, `TargetArn`, or `TopicArn` must be provided.

**Parameters**

- **Message** – The message you want to send.
- **MessageAttributes** – Message attributes for Publish action.
- **MessageStructure** – Set MessageStructure to json if you want to send a different message for each protocol.
- **Subject** – Optional parameter to be used as the "Subject" line when the message is delivered to email endpoints. This field will also be included, if present, in the standard JSON messages delivered to other endpoints.
- **PhoneNumber** – The phone number to which you want to deliver an SMS message. Use E.164 format.
- **TargetArn** – If you don't specify a value for the TargetArn parameter, you must specify a value for the PhoneNumber or TopicArn parameters.
- **TopicArn** – The topic you want to publish to.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: `''`)
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (`JsonPath`) – The portion of the state input data to be used as input for the state (default: `JsonPath(path=Root())`)
- **OutputPath** (`JsonPath`) – The portion of the state input data to be passed to the next state (default: `JsonPath(path=Root())`)
- **ResultPath** (`JsonPath`) – Where in the state input data to place the results of this state (default: `JsonPath(path=Root())`)
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run

- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

### Amazon SQS

Amazon SQS Task state.

**class** rhodes.states.services.sqs.**AmazonSqs**(*title*, *, *Comment=None*, *DelaySeconds=None*, *MessageAttribute=None*, *MessageBody=None*, *MessageDeduplicationId=None*, *MessageGroupId=None*, *QueueUrl=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPONSE: ">*)

Bases: *rhodes.states.State*

Deliver a message to the specified queue.

See service docs for more details.

**Parameters**

- **DelaySeconds** – The length of time, in seconds, for which to delay a specific message. Valid values: 0 to 900. Maximum: 15 minutes.
- **MessageAttribute** – Each message attribute consists of a Name, Type, and Value.
- **MessageBody** – The message to send. The maximum string size is 256 KB.
- **MessageDeduplicationId** – The token used for deduplication of sent messages.
- **MessageGroupId** – The tag that specifies that a message belongs to a specific message group.

- **QueueUrl** – The URL of the Amazon SQS queue to which a message is sent.
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a Pass state after this state that promotes a path in the input to this state's ResultPath.

Path *must* start with a path relative this state's ResultPath as indicated by a @. prefix.

**Parameters** **path** (Union[str, Enum, JSONPath, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** Dict

### AWS Step Functions

AWS Step Functions Task state.

**class** rhodes.states.services.stepfunctions.**AwsStepFunctions**(*title*, ***, *Comment=None*, *StateMachineArn=None*, *Input=None*, *Next=None*, *End=None*, *InputPath=JsonPath(path=Root())*, *OutputPath=JsonPath(path=Root())*, *ResultPath=JsonPath(path=Root())*, *Retry=None*, *Catch=None*, *TimeoutSeconds=None*, *HeartbeatSeconds=None*, *Pattern=<IntegrationPattern.REQUEST_RESPO ">*)

Bases: *rhodes.states.State*

Start a state machine execution.

See service docs for more details.

**Parameters**

- **StateMachineArn** (JsonPath, AWSHelperFn, str, or Enum) – The AWS Step Functions state machine to invoke
- **Input** (Parameters, JsonPath, AWSHelperFn, dict, str, or Enum) – Data to provide to the state machine as input
- **title** (*str*) – Name of state in state machine
- **Comment** (*str*) – Human-readable description of the state (default: '')
- **Next** – The state that will follow this state
- **End** (*bool*) – This state is a terminal state
- **InputPath** (JsonPath) – The portion of the state input data to be used as input for the state (default: JsonPath(path=Root()))
- **OutputPath** (JsonPath) – The portion of the state input data to be passed to the next state (default: JsonPath(path=Root()))
- **ResultPath** (JsonPath) – Where in the state input data to place the results of this state (default: JsonPath(path=Root()))
- **Retry** –
- **Catch** –
- **TimeoutSeconds** (*int*) – Maximum time that this state is allowed to run
- **HeartbeatSeconds** (*int*) – Maximum time allowed between heartbeat responses from state
- **Pattern** (IntegrationPattern) – Step Functions integration pattern (default: <IntegrationPattern.REQUEST_RESPONSE: ''>)

**end**()
Make this state a terminal state.

**member_of = None**

**promote**(*path*)
Add a `Pass` state after this state that promotes a path in the input to this state's `ResultPath`.

Path *must* start with a path relative this state's `ResultPath` as indicated by a `@.` prefix.

**Parameters** **path** (`Union`[`str`, `Enum`, `JSONPath`, *JsonPath*]) – Path to promote

**Return type** *Pass*

**then**(*next_state*)
Set the next state in this state machine.

**to_dict**() → Dict[KT, VT]
Serialize state as a dictionary.

**Return type** `Dict`

## 5.2 choice_rules

Rules for defining the branching logic in `Choice` states.

See Step Functions docs for more details.

**class** rhodes.choice_rules.**VariablePath**(*path*)
Bases: *rhodes.structures.JsonPath*

`JsonPath` variant with overloading helper methods to generate choice rules.

**class** rhodes.choice_rules.**ChoiceRule**
Bases: `object`

Base class for all choice rules.

**member_of = None**

**to_dict**()
Serialize state as a dictionary.

**then**(*state*)

**class** rhodes.choice_rules.**StringEquals**(*, *Variable=None*, *Next=None*, *Value=None*)
Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*str*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**StringLessThan**(*, *Variable=None*, *Next=None*, *Value=None*)
Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated

- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*str*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**StringGreaterThan**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*str*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**StringLessThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*str*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**StringGreaterThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*str*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**NumericEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**NumericLessThan**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true

- **Value** – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**NumericGreaterThan**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**NumericLessThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**NumericGreaterThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**BooleanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*bool*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**TimestampEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (`VariablePath`) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*datetime*) – The value to which to compare `Variable`

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**TimestampLessThan**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (VariablePath) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*datetime*) – The value to which to compare Variable

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**TimestampGreaterThan**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (VariablePath) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*datetime*) – The value to which to compare Variable

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**TimestampLessThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (VariablePath) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*datetime*) – The value to which to compare Variable

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**TimestampGreaterThanEquals**(*, *Variable=None*, *Next=None*, *Value=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

**Parameters**

- **Variable** (VariablePath) – Path to value in state input that will be evaluated
- **Next** – The state to which to continue if this rule evaluates as true
- **Value** (*datetime*) – The value to which to compare Variable

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**And**(*, *Rules=None*, *Next=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

Matches only if all of the provided rules are true. :param Rules: One or more *ChoiceRule* to evaluate for this rule :param Next: The state to which to continue if this rule evaluates as true

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**Or**(*, *Rules=None*, *Next=None*)

Bases: *rhodes.choice_rules.ChoiceRule*

Matches if any of the provided rules are true. :param Rules: One or more *ChoiceRule* to evaluate for this rule :param Next: The state to which to continue if this rule evaluates as true

**to_dict**(*suppress_next=False*)

**class** rhodes.choice_rules.**Not**(*, *Rule=None*, *Next=None*)
Bases: *rhodes.choice_rules.ChoiceRule*

Matches only if the provided rule is false.

**Parameters**

- **Rule** (ChoiceRule) – Rule that must evaluate as false
- **Next** – The state to which to continue if this rule evaluates as true

**to_dict**(*suppress_next=False*)
Serialize state as a dictionary.

rhodes.choice_rules.**all_**(**rules*)
Helper to assemble several rules into an *And* rule.

**Return type** *And*

rhodes.choice_rules.**any_**(**rules*)
Helper to assemble several rules into an *Or* rule.

**Return type** *Or*

## 5.3 structures

Helper structures for Rhodes.

**class** rhodes.structures.**JsonPath**(*path*)
Bases: object

Represents a JSONPath variable in request/response body.

**Parameters** **path** – JSONPath to desired data in state input data

**to_dict**()
Serialize path for use in serialized state machine definition.

**Return type** str

**class** rhodes.structures.**ContextPath**(*path='$$'*)
Bases: object

Represents a JSONPath(ish) variable in the Context Object.

**Parameters** **path** (*str*) – Path to value in Context Object.

In addition to specifying the path manually, you can specify valid paths through dot-notation. For example:

- `ContextPath("$$.Execution.Id") == ContextPath().Execution.Id`
- `ContextPath("$$.Map.Item.Value.foo.bar") == ContextPath().Map.Item.Value.foo.bar`

**to_dict**()
Serialize path for use in serialized state machine definition.

**Return type** str

**class** rhodes.structures.**Parameters**(***kwargs*)
Bases: object

Represents parameters that can be passed to various state fields.

If a *JsonPath* is provided as a field value, the field name will be automatically appended with a `.$` value when the state machine is serialized to indicate to Step Functions to de-reference the value.

For example:

```
>>> Parameters(foo=bar, baz=JsonPath("$.wat.wow")).to_dict()
{"foo": "bar", "baz.$": "$.wat.wow"}
```

**to_dict**()
Serialize parameters for use in serialized state machine definition.

**Return type** Dict[str, Any]

## 5.4 identifiers

Static identifiers used within Rhodes.

**class** rhodes.identifiers.**ServiceArn**
Bases: enum.Enum

Step Functions service integrations ARNs.

Used as `Resource` value when creating a `Task` that uses one of these service integrations.

**AWSLAMBDA = 'arn:aws:states:::lambda:invoke'**

**BATCH = 'arn:aws:states:::batch:submitJob'**

**ECS = 'arn:aws:states:::ecs:runTask'**

**SNS = 'arn:aws:states:::sns:publish'**

**SQS = 'arn:aws:states:::sqs:sendMessage'**

**GLUE = 'arn:aws:states:::glue:startJobRun'**

**STEP_FUNCTIONS = 'arn:aws:states:::states:startExecution'**

**DYNAMODB_GET_ITEM = 'arn:aws:states:::dynamodb:getItem'**

**DYNAMODB_PUT_ITEM = 'arn:aws:states:::dynamodb:putItem'**

**DYNAMODB_DELETE_ITEM = 'arn:aws:states:::dynamodb:deleteItem'**

**DYNAMODB_UPDATE_ITEM = 'arn:aws:states:::dynamodb:updateItem'**

**SAGEMAKER_CREATE_ENDPOINT = 'arn:aws:states:::sagemaker:createEndpoint'**

**SAGEMAKER_CREATE_ENDPOINT_CONFIG = 'arn:aws:states:::sagemaker:createEndpointConfig'**

**SAGEMAKER_CREATE_HYPER_PARAMETER_TUNING_JOB = 'arn:aws:states:::sagemaker:createHyperP**

**SAGEMAKER_CREATE_LABELING_JOB = 'arn:aws:states:::sagemaker:createLabelingJob'**

**SAGEMAKER_CREATE_MODEL = 'arn:aws:states:::sagemaker:createModel'**

**SAGEMAKER_CREATE_TRAINING_JOB = 'arn:aws:states:::sagemaker:createTrainingJob'**

**SAGEMAKER_CREATE_TRANSFORM_JOB = 'arn:aws:states:::sagemaker:createTransformJob'**

**SAGEMAKER_UPDATE_ENDPOINT = 'arn:aws:states:::sagemaker:updateEndpoint'**

**class** rhodes.identifiers.**IntegrationPattern**
Bases: enum.Enum

Service integration pattern types.

Used either when building a service integration `Task` manually or when configuring a `ServiceIntegration` helper class.

**REQUEST_RESPONSE = ''**

**SYNCHRONOUS = '.sync'**

**WAIT_FOR_CALLBACK = '.waitForTaskToken'**

## 5.5 exceptions

Exceptions for use in Rhodes.

**exception** rhodes.exceptions.**RhodesError**
Bases: Exception

Common base for all Rhodes exceptions.

**exception** rhodes.exceptions.**IncompleteDefinitionError**
Bases: *rhodes.exceptions.RhodesError*

Raise when an incomplete state machine definition is found.

**exception** rhodes.exceptions.**InvalidDefinitionError**
Bases: *rhodes.exceptions.RhodesError*

Raised when an invalid state machine definition is found.

# Python Module Index

## r

# Index

## F



## G



## I



## J



## M

## N



## O



## P



## R



## S

## T

## U



## V



## W