TaskFlow

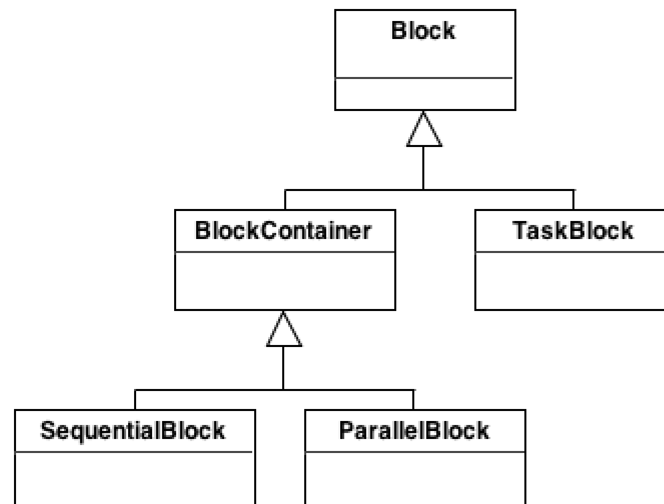
What is the Flow?
Flow is a set of tasks that describes an OpenStack command and should be executed as a transaction.

Task
Task is an primitive OpenStack operation that retrieves some data or makes changes in OpenStack component state, can be executed or reverted.

Task(object)	**Base class for all tasks.**
__init__(name)	Constructor. Accepts task name.
execute(**kwargs)	Executes the task.
revert(**kwargs)	Reverts the task.

The main goal of the TaskFlow is to group tasks into one set and execute it as a transaction.

Block
Block is a primitive used to describe a Flow pattern. There are different types of blocks that allows us to describe different behaviour and control the Flow execution.



Block(object)	**Base class for all blocks.**
trigger(callback_func)	Registers a callback trigger called on entering the block
postTrigger(callback_func)	Registers a callback trigger called on leaving the block
revertTrigger(callback_func)	Registers a callback trigger called on entering the block during the revert
postRevertTrigger(callback_func)	Registers a callback trigger called on leaving the block during the revert
bool hasTriggers()	Returns True if the block has registered triggers
@abc.abstractmethod acceptVisitor(visitor)	Abstract method. Accepts visitor through the blocks hierarchy

BlockContainer(Block)	**Base class for Blocks that can contain nested Blocks.**
append(block)	Appends block to the current block.

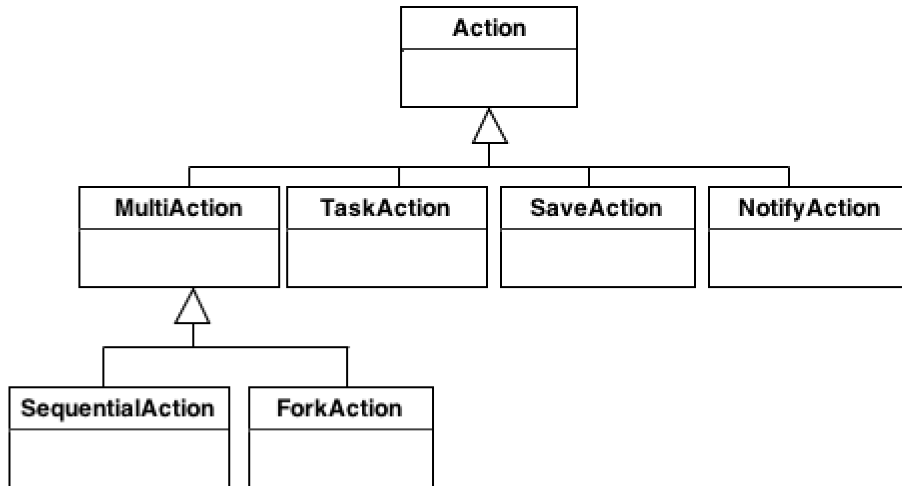
SequentialBlock(BlockContainer)	**Contains blocks that should be executed sequentially.**

ParallelBlock(BlockContainer)	**Contains blocks that should be executed in parallel.**

TaskBlock(Block)	**Contains a Task that should be executed, saves data produced by the Task, notifies about Task completion.**
__init__(task_class, task_args=[])	Constructor. Accepts task class object that will be executed and arguments that will be passed to task constructor.
save(param_name, method_name)	Tells us that data returned from task method ‘method_name’ will be saved to the common flow dictionary with key ‘param_name’. Other tasks can use this data later.

Action

Action is a primitive graph node that do a progress of the Flow. Each action knows how to execute and revert itself.



Action(object)	**Base class for all actions.**
__init__(flow)	Constructor. Accepts flow object.
@abc.abstractmethod *execute()*	Abstract method. Executes an action.
@abc.abstractmethod *revert()*	Abstract method. Reverts an action.

MultiAction(Action)	**Base class for actions that contain nested actions.**
__init__(flow)	Constructor. Accepts flow object.
action(action)	Adds child Action.

SequentialAction(MultiAction)	**Executes child actions sequentially. Stops when some action fails. Saves actions that were executed. Reverts executed actions.**
__init__(flow)	Constructor. Accepts flow object.
execute()	Executes child actions sequentially.

revert()	Reverts executed actions.

ForkAction(MultiAction)	**Executes child actions in parallel. Creates new thread for each action. Threads pool can be added to the Flow object to control a number of runned threads. Saves actions that were executed. Reverts executed actions.**
__init__(flow)	Constructor. Accepts flow object.
execute()	Executes child actions in parallel.
revert()	Reverts executed actions.
threadCompleted(thread, status)	Callback method for ThreadRunner. Accepts thread and execution status. ThreadRunner calls this method after thread's completion.

TaskAction(Action)	**Executes and reverts given Task object. Notifies about starting and finishing of execution or reverting. Marks Flow as failed if Task throws an exception.**
__init__(flow, task)	Constructor. Accepts flow object and Task object.
execute()	Executes the Task.
revert()	Reverts the Task.

SaveAction(Action)	**Saves data generated after the Task execution to the common Flow dictionary. Another tasks can access this data from the dictionary.**
__init__(flow, task, param_name, method_name)	Constructor. Accepts flow object and Task object. 'parameter_name' is a key value to find the saved data. 'method_name' is a name of the method of task object that returns the data.
execute()	Obtains the Task data. Saves a pair ('param_name', data) to the Flow dictionary.
revert()	Removes data from the dictionary.

Notify(Action)	**Notifies listeners on entering or leaving the action.**
__init__(flow, action)	Constructor. Accepts flow object and action that has registered listeners.
execute()	Notifies about entering the action. Executes the action. Notifies about leaving the action.
revert()	Notifies about reverting the action. Reverts the action. Notifies that action was reverted.

Now we have primitives to describe and execute the Flow. There are Blocks and Actions. The next step is to build the actions graph using the pattern described with the Blocks. The following table shows us how Blocks description corresponds to Actions.

SequentialBlock ->	SequentialAction
ParallelBlock ->	ForkAction
.trigger ->	NotifyAction
.postTrigger ->	NotifyAction
.revertTrigger ->	NotifyAction
.postRevertTrigger ->	NotifyAction
TaskBlock ->	TaskAction
.save ->	SaveAction

FlowBuilder
FlowBuilder is used to obtain a graph of actions from the blocks pattern. FlowBuilder accepts the blocks pattern and returns a Flow object that contains a graph of actions. FlowBuilder uses an actions factory to create actions.

Why do we need a factory?
ActionsFactory creates an action objects. By default it creates an actions described above. But what if the user wants to use his own action that implements some specific behaviour? She

simply can override the factory method with another one that will create her action object. ActionsFactory allows user to be more flexible with changing a Flow behavior using the same Flow description (blocks).

What is the difference between Task and Action?
Task is a primitive OpenStack operation. Action is a primitive node that allows to order and control execution of Tasks.

Flow

Flow contains a graph of Actions. Flow runs as a transaction and manages execution of actions. Flow can execute and revert actions, notify about changes on it's state, etc. Flow can be interrupted, resumed or cancelled.

Flow(object)	**Runs and manages tasks execution.**
__inti__(uuid)	Constructor. Accepts flow uuid.
run()	Runs tasks in the defined order as a transaction. Reverts transaction if it fails.
onTaskStarted(task)	Callback method. Notifies about the task execution started.
onTaskFinished(task, state)	Callback method. Notifies about the task completion and its state.
onTaskRevertStarted(task)	Callback method. Notifies that the task reverting started.
onTaskReverted(task, state, exception)	Callback method. Notifies that the task was reverted of reverting was failed.
self.state	Current state of the flow.
self.kwargs	Dictionary of data produced by the tasks.

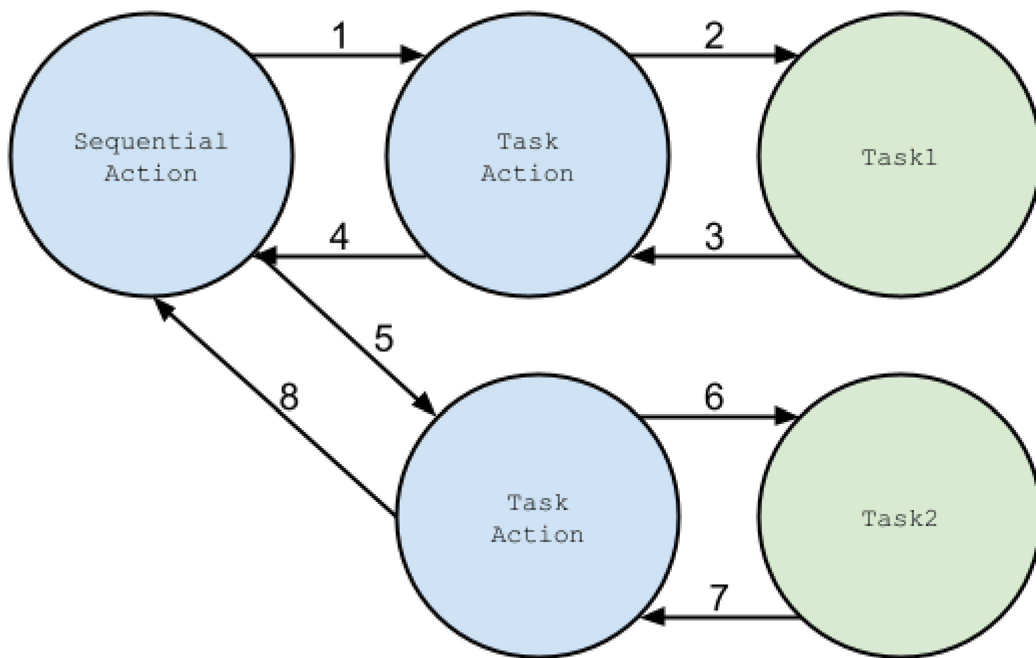
How is data passed between tasks?
TaskBlock has a save() method that accept a task method name and a key. The data returned by the given task method saves to the Flow kwargs dictionary with the given key. Task execute() and revert() methods accept Flow kwargs as a parameter.

Examples:
Following examples show how flow description corresponds to actions graph. Blue circles are Actions, green circles are Tasks. Arrows show the execution path of the graph.

1. Sequential block contains two Task blocks. Tasks will be executed sequentially.

```
(SequentialBlock()
  .append(TaskBlock(Task1))
  .append(TaskBlock(Task2)))
```

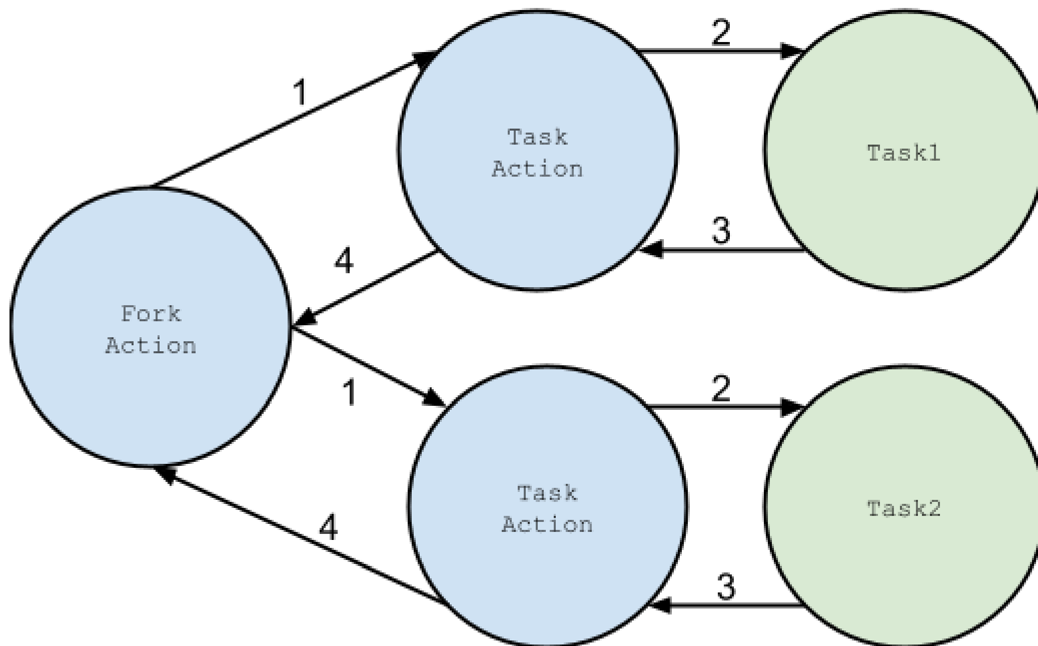
Actions graph:



2. Parallel block contains two Task blocks. Tasks will be executed in parallel.

```
(ParallelBlock()
  .append(TaskBlock(Task1))
  .append(TaskBlock(Task2)))
```

Actions graph:



3. Task block with triggers and save.

```
(TaskBlock(Task1)
  .trigger(Callback1)
  .postTrigger(Callback2)
  .save('key', 'getValue'))
```

Actions graph: