`BaseSwarm` Documentation

Table of Contents

---

1. Introduction

The Swarms library is designed to provide a framework for swarm simulation architectures. Swarms are collections of autonomous agents or workers that collaborate to perform tasks and achieve common goals. This documentation will guide you through the functionality and usage of the Swarms library, explaining the purpose and implementation details of the provided classes and methods.

2. Class Definition

`BaseSwarm` Class

The `BaseSwarm` class is an abstract base class that serves as the foundation for swarm simulation architectures. It defines the core functionality and methods required to manage and interact with a swarm of workers.

```python
from abc import ABC, abstractmethod
from typing import List

from swarms.swarms.base import AbstractWorker


class BaseSwarm(ABC):
    """
    Abstract class for swarm simulation architectures

    Methods:
    ---------
    ...
    """

    # The class definition and constructor are provided here.

    @abstractmethod
    def __init__(self, workers: List["AbstractWorker"]):
        """Initialize the swarm with workers"""

    # Other abstract methods are listed here.
```

3. Methods

`communicate()`

The `communicate()` method allows the swarm to exchange information through the orchestrator, protocols, and the universal communication layer.

Usage Example 1:

```python
swarm = YourSwarmClass(workers)
swarm.communicate()
```

Usage Example 2:

```python
# Another example of using the communicate method
swarm = YourSwarmClass(workers)
swarm.communicate()
```

`run()`

The `run()` method executes the swarm, initiating its activities.

Usage Example 1:

```python
swarm = YourSwarmClass(workers)
swarm.run()
```

Usage Example 2:

```python
# Another example of running the swarm
swarm = YourSwarmClass(workers)
swarm.run()
```

`arun()`

The `arun()` method runs the swarm asynchronously, allowing for parallel execution of tasks.

Usage Example 1:

```python
swarm = YourSwarmClass(workers)
swarm.arun()
```

Usage Example 2:

```python
# Another example of running the swarm asynchronously
swarm = YourSwarmClass(workers)
swarm.arun()
```

`add_worker(worker: "AbstractWorker")`

The `add_worker()` method adds a worker to the swarm.

Parameters:
- `worker` (AbstractWorker): The worker to be added to the swarm.

Usage Example:

```python
swarm = YourSwarmClass([])
worker = YourWorkerClass()
swarm.add_worker(worker)
```

`remove_worker(worker: "AbstractWorker")`

The `remove_worker()` method removes a worker from the swarm.

Parameters:

- `worker` (AbstractWorker): The worker to be removed from the swarm.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_id("worker_id")
swarm.remove_worker(worker)
```

`broadcast(message: str, sender: Optional["AbstractWorker"] = None)`

The `broadcast()` method sends a message to all workers in the swarm.

Parameters:

- `message` (str): The message to be broadcasted.
- `sender` (Optional[AbstractWorker]): The sender of the message (optional).

Usage Example 1:

```python
swarm = YourSwarmClass(workers)
message = "Hello, everyone!"
```

swarm.broadcast(message)

```

**Usage Example 2:**

```python
# Another example of broadcasting a message
swarm = YourSwarmClass(workers)
message = "Important announcement!"
sender = swarm.get_worker_by_name("Supervisor")
swarm.broadcast(message, sender)
```

`reset()`

The `reset()` method resets the swarm to its initial state.

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm.reset()
```

`plan(task: str)`
```

The `plan()` method instructs workers to individually plan using a workflow or pipeline for a specified task.

Parameters:

- `task` (str): The task for which workers should plan.

Usage Example:

```python
swarm = YourSwarmClass(workers)
task = "Perform data analysis"
swarm.plan(task)
```

`direct_message(message: str, sender: "AbstractWorker", recipient: "AbstractWorker")`

The `direct_message()` method sends a direct message from one worker to another.

Parameters:

- `message` (str): The message to be sent.
- `sender` (AbstractWorker): The sender of the message.
- `recipient` (AbstractWorker): The recipient of the message.

Usage Example:

```python
swarm = YourSwarmClass(workers)
sender = swarm.get_worker_by_name("Worker1")
recipient = swarm.get_worker_by_name("Worker2")
message = "Hello

, Worker2!"
swarm.direct_message(message, sender, recipient)
```

`autoscaler(num_workers: int, worker: List["AbstractWorker"])`

The `autoscaler()` method acts as an autoscaler, dynamically adjusting the number of workers based on system load or other criteria.

Parameters:
- `num_workers` (int): The desired number of workers.
- `worker` (List[AbstractWorker]): A list of workers to be managed by the autoscaler.

Usage Example:

```python
swarm = YourSwarmClass([])
workers = [YourWorkerClass() for _ in range(10)]
swarm.autoscaler(5, workers)
```

`get_worker_by_id(id: str) -> "AbstractWorker"`

The `get_worker_by_id()` method locates a worker in the swarm by their ID.

Parameters:

- `id` (str): The ID of the worker to locate.

Returns:

- `AbstractWorker`: The worker with the specified ID.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker_id = "worker_123"
worker = swarm.get_worker_by_id(worker_id)
```

`get_worker_by_name(name: str) -> "AbstractWorker"`

The `get_worker_by_name()` method locates a worker in the swarm by their name.

Parameters:

- `name` (str): The name of the worker to locate.

Returns:

- `AbstractWorker`: The worker with the specified name.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker_name = "Alice"
worker = swarm.get_worker_by_name(worker_name)
```

`assign_task(worker: "AbstractWorker", task: Any) -> Dict`

The `assign_task()` method assigns a task to a specific worker.

Parameters:

- `worker` (AbstractWorker): The worker to whom the task should be assigned.
- `task` (Any): The task to be assigned.

Returns:

- `Dict`: A dictionary indicating the status of the task assignment.

Usage Example:

```python
swarm = YourSwarmClass(workers)
```

```
worker = swarm.get_worker_by_name("Worker1")
task = "Perform data analysis"
result = swarm.assign_task(worker, task)
```

`get_all_tasks(worker: "AbstractWorker", task: Any)`

The `get_all_tasks()` method retrieves all tasks assigned to a specific worker.

Parameters:
- `worker` (AbstractWorker): The worker for whom tasks should be retrieved.
- `task` (Any): The task to be retrieved.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_name("Worker1")
tasks = swarm.get_all_tasks(worker, "data analysis")
```

`get_finished_tasks() -> List[Dict]`

The `get_finished_tasks()` method retrieves all tasks that have been completed by the workers in the swarm.

Returns:

- `List[Dict]`: A list of dictionaries representing finished tasks.

Usage Example:

```python
swarm = YourSwarmClass(workers)
finished_tasks = swarm.get_finished_tasks()
```

`get_pending_tasks() -> List[Dict]`

The `get_pending_tasks()` method retrieves all tasks that are pending or yet to be completed by the workers in the swarm.

Returns:

- `List[Dict]`: A list of dictionaries representing pending tasks.

Usage Example:

```python
swarm = YourSwarmClass(workers)
pending_tasks = swarm.get_pending_tasks()
```

`pause_worker(worker: "AbstractWorker", worker_id: str)`

The `pause_worker()` method pauses a specific worker, temporarily suspending their activities.

Parameters:

- `worker` (AbstractWorker): The worker to be paused.
- `worker_id` (str): The ID of the worker to be paused.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_name("Worker1")
worker_id = "worker_123"
swarm.pause_worker(worker, worker_id)
```

`resume_worker(worker: "AbstractWorker", worker_id: str)`

The `resume_worker()` method resumes a paused worker, allowing them to continue their activities.

Parameters:

- `worker` (AbstractWorker): The worker to be resumed.
- `worker_id` (str): The ID of the worker to be resumed.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_name("Worker1")
worker_id = "worker_123"
swarm.resume_worker(worker, worker_id)
```

`stop_worker(worker: "AbstractWorker", worker_id: str)`

The `stop_worker()` method stops a specific worker, terminating their activities.

Parameters:

- `worker` (AbstractWorker): The worker to be stopped.
- `worker_id` (str): The ID of the worker to be stopped.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_name("Worker1")
worker_id = "worker_123"
swarm.stop_worker(worker, worker_id)
```

`restart_worker(worker: "AbstractWorker")`

The `restart_worker()` method restarts a worker, resetting them to their initial state.

Parameters:

- `worker` (AbstractWorker): The worker to be restarted.

Usage Example:

```python
swarm = YourSwarmClass(workers)
worker = swarm.get_worker_by_name("Worker1")
swarm.restart_worker(worker)
```

`scale_up(num_worker: int)`

The `scale_up()` method increases the number of workers in the swarm.

Parameters:

- `num_worker` (int): The number of workers to add to the swarm.

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm.scale_up(5)
```

`scale_down(num_worker: int)`

The `scale_down()` method decreases the number of workers in the swarm.

Parameters:

- `num_worker` (int): The number of workers to remove from the swarm.

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm.scale_down(3)
```

`scale_to(num_worker: int)`

The `scale_to()` method scales the swarm to a specific number of workers.

Parameters:

- `num_worker` (int): The desired number of workers.

Usage Example:

```python
swarm = YourSwarmClass(workers)
```

swarm.scale_to(10)

```

### `get

_all_workers() -> List["AbstractWorker"]` <a name="get_all_workers"></a>

The `get_all_workers()` method retrieves a list of all workers in the swarm.

**Returns:**

- `List[AbstractWorker]`: A list of all workers in the swarm.

**Usage Example:**

```python

swarm = YourSwarmClass(workers)

all_workers = swarm.get_all_workers()

```

`get_swarm_size() -> int`

The `get_swarm_size()` method returns the size of the swarm, which is the total number of workers.

Returns:

- `int`: The size of the swarm.
```

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm_size = swarm.get_swarm_size()
```

`get_swarm_status() -> Dict`

The `get_swarm_status()` method provides information about the current status of the swarm.

Returns:

- `Dict`: A dictionary containing various status indicators for the swarm.

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm_status = swarm.get_swarm_status()
```

`save_swarm_state()`

The `save_swarm_state()` method allows you to save the current state of the swarm, including worker configurations and task assignments.

Usage Example:

```python
swarm = YourSwarmClass(workers)
swarm.save_swarm_state()
```

This comprehensive documentation covers the Swarms library, including the `BaseSwarm` class and its methods. You can use this documentation as a guide to understanding and effectively utilizing the Swarms framework for swarm simulation architectures. Feel free to explore further and adapt the library to your specific use cases.