AgentRegistry Documentation

The `AgentRegistry` class is designed to manage a collection of agents, providing methods for adding, deleting, updating, and querying agents. This class ensures thread-safe operations on the registry, making it suitable for concurrent environments. Additionally, the `AgentModel` class is a Pydantic model used for validating and storing agent information.

Attributes

AgentModel

Attribute	Type	Description
`agent_id`	`str`	The unique identifier for the agent.
`agent`	`Agent`	The agent object.

AgentRegistry

Attribute	Type	Description
`agents`	`Dict[str, AgentModel]`	A dictionary mapping agent IDs to `AgentModel` instances.
`lock`	`Lock`	A threading lock for thread-safe operations.

Methods

`__init__(self)`

Initializes the `AgentRegistry` object.

- **Usage Example:**

  ```python
  registry = AgentRegistry()
  ```

`add(self, agent_id: str, agent: Agent) -> None`

Adds a new agent to the registry.

- **Parameters:**
  - `agent_id` (`str`): The unique identifier for the agent.
  - `agent` (`Agent`): The agent to add.

- **Raises:**
  - `ValueError`: If the agent ID already exists in the registry.
  - `ValidationError`: If the input data is invalid.

- **Usage Example:**

  ```python
  agent = Agent(agent_name="Agent1")
  registry.add("agent_1", agent)
  ```

`delete(self, agent_id: str) -> None`

Deletes an agent from the registry.

- **Parameters:**
  - `agent_id` (`str`): The unique identifier for the agent to delete.

- **Raises:**
  - `KeyError`: If the agent ID does not exist in the registry.

- **Usage Example:**
  ```python
  registry.delete("agent_1")
  ```

`update_agent(self, agent_id: str, new_agent: Agent) -> None`

Updates an existing agent in the registry.

- **Parameters:**
  - `agent_id` (`str`): The unique identifier for the agent to update.
  - `new_agent` (`Agent`): The new agent to replace the existing one.

- **Raises:**
  - `KeyError`: If the agent ID does not exist in the registry.
  - `ValidationError`: If the input data is invalid.

- **Usage Example:**

  ```python

  new_agent = Agent(agent_name="UpdatedAgent")

  registry.update_agent("agent_1", new_agent)

  ```

`get(self, agent_id: str) -> Agent`

Retrieves an agent from the registry.

- **Parameters:**

  - `agent_id` (`str`): The unique identifier for the agent to retrieve.

- **Returns:**

  - `Agent`: The agent associated with the given agent ID.

- **Raises:**

  - `KeyError`: If the agent ID does not exist in the registry.

- **Usage Example:**

  ```python

  agent = registry.get("agent_1")

  ```

`list_agents(self) -> List[str]`

Lists all agent identifiers in the registry.

- **Returns:**
  - `List[str]`: A list of all agent identifiers.

- **Usage Example:**
  ```python
  agent_ids = registry.list_agents()
  ```

`query(self, condition: Optional[Callable[[Agent], bool]] = None) -> List[Agent]`

Queries agents based on a condition.

- **Parameters:**
  - `condition` (`Optional[Callable[[Agent], bool]]`): A function that takes an agent and returns a boolean indicating whether the agent meets the condition. Defaults to `None`.

- **Returns:**
  - `List[Agent]`: A list of agents that meet the condition.

- **Usage Example:**
  ```python
  def is_active(agent):
      return agent.is_active
  ```

```
active_agents = registry.query(is_active)
```

`find_agent_by_name(self, agent_name: str) -> Agent`

Finds an agent by its name.

- **Parameters:**
  - `agent_name` (`str`): The name of the agent to find.

- **Returns:**
  - `Agent`: The agent with the specified name.

- **Usage Example:**
  ```python
  agent = registry.find_agent_by_name("Agent1")
  ```

Full Example

```python
from swarms.structs.agent_registry import AgentRegistry
from swarms import Agent, OpenAIChat, Anthropic
```

```
# Initialize the agents
growth_agent1 = Agent(
  agent_name="Marketing Specialist",
   system_prompt="You're the marketing specialist, your purpose is to help companies grow by improving their marketing strategies!",
  agent_description="Improve a company's marketing strategies!",
  llm=OpenAIChat(),
  max_loops="auto",
  autosave=True,
  dashboard=False,
  verbose=True,
  streaming_on=True,
  saved_state_path="marketing_specialist.json",
  stopping_token="Stop!",
  interactive=True,
  context_length=1000,
)

growth_agent2 = Agent(
  agent_name="Sales Specialist",
  system_prompt="You're the sales specialist, your purpose is to help companies grow by improving their sales strategies!",
  agent_description="Improve a company's sales strategies!",
  llm=Anthropic(),
  max_loops="auto",
  autosave=True,
```

```
  dashboard=False,
  verbose=True,
  streaming_on=True,
  saved_state_path="sales_specialist.json",
  stopping_token="Stop!",
  interactive=True,
  context_length=1000,
)

growth_agent3 = Agent(
  agent_name="Product Development Specialist",
   system_prompt="You're the product development specialist, your purpose is to help companies grow by improving their product development strategies!",
  agent_description="Improve a company's product development strategies!",
  llm=Anthropic(),
  max_loops="auto",
  autosave=True,
  dashboard=False,
  verbose=True,
  streaming_on=True,
  saved_state_path="product_development_specialist.json",
  stopping_token="Stop!",
  interactive=True,
  context_length=1000,
)
```

```
growth_agent4 = Agent(
  agent_name="Customer Service Specialist",
   system_prompt="You're the customer service specialist, your purpose is to help companies grow by improving their customer service strategies!",
  agent_description="Improve a company's customer service strategies!",
  llm=OpenAIChat(),
  max_loops="auto",
  autosave=True,
  dashboard=False,
  verbose=True,
  streaming_on=True,
  saved_state_path="customer_service_specialist.json",
  stopping_token="Stop!",
  interactive=True,
  context_length=1000,
)



# Register the agents\
registry = AgentRegistry()

# Register the agents
registry.add("Marketing Specialist", growth_agent1)
registry.add("Sales Specialist", growth_agent2)
registry.add("Product Development Specialist", growth_agent3)
registry.add("Customer Service Specialist", growth_agent4)
```

```

## Logging and Error Handling

Each method in the `AgentRegistry` class includes logging to track the execution flow and captures errors to provide detailed information in case of failures. This is crucial for debugging and ensuring smooth operation of the registry. The `report_error` function is used for reporting exceptions that occur during method execution.

## Additional Tips

- Ensure that agents provided to the `AgentRegistry` are properly initialized and configured to handle the tasks they will receive.
- Utilize the logging information to monitor and debug the registry operations.
- Use the `lock` attribute to ensure thread-safe operations when accessing or modifying the registry.
```