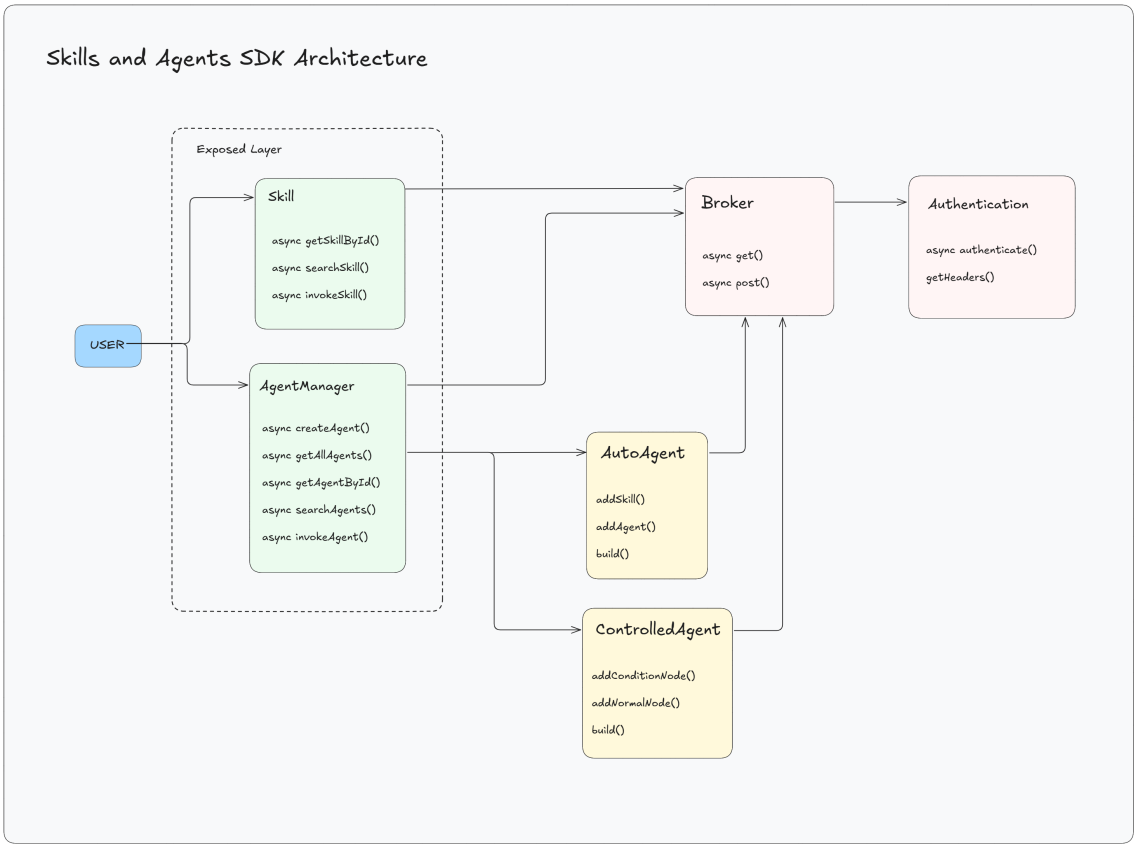
# Agents-Skill SDK Documentation

## Overview

The Agents-Skill SDK provides a structured interface for managing and interacting with AI agents and skills. It facilitates the creation, configuration, and invocation of agents and skills through a set of well-defined classes and methods. The SDK is built on top of a broker that handles HTTP communication with the backend services.




## Table of Contents


1. **User Exposed Classes**
    - Skill Class
    - AgentManager Class
2. **AgentManager Subclasses**
    - ControlledAgent
    - AutoAgent
3. **Utilities**
    - Authentication
    - Broker

# 1. User Exposed Classes

## Skill Class

Interacts with skills via the broker.

**Key Methods:**

- **`get_skill_by_id(skill_id: str, version: str) -> dict`**
  Fetches skill details by skill ID.

  **Mandatory Parameters:**

  - `skill_id` (str): The ID of the skill to fetch.
  - `version` (str): The version of the skill.

  **Functionality:**
  This method retrieves the details of a skill by its ID and the specified version. The response will contain the full details of the skill for further interaction or examination.

---

- **`search_skills(query: str, skip=0, limit=50) -> dict`**
  Searches for skills based on a query.

  **Mandatory Parameters:**

  - `query` (str): The search query.

  **Optional Parameters:**

  - `skip` (int): The number of items to skip for pagination. Default is `0`.
  - `limit` (int): The number of items to return. Default is `50`.

  **Functionality:**
  This method allows you to search for skills using a query string. Pagination can be controlled using the `skip` and `limit` parameters, which specify how many items to skip and the maximum number of results to return, respectively.

---

- **`invoke_skill_by_id(context: Dict[str, Any], event: Dict[str, Any], skill_id: str, version='1', **kwargs) -> dict`**
  Invokes a skill by its ID with the provided context and event.

  **Mandatory Parameters:**

  - `context` (dict): The context for the skill invocation.
  - `event` (dict): The event triggering the skill invocation.
  - `skill_id` (str): The ID of the skill to invoke.

  **Optional Parameters:**

  - `version` (str): The version of the skill. Default is `'1'`.
  - `kwargs`: Additional keyword arguments for the request. These can be used to provide extra parameters or overrides for the invocation.

  **Functionality of `kwargs`:**
  `kwargs` allows you to pass additional data to the skill invocation. This can be extra parameters not defined in the main function signature (e.g., custom

configurations, parameters specific to the skill). These are included in the body of the request, allowing for more flexible skill invocation.

---

- **`get_agent_args_by_skills(skills: list) -> dict`**
  Fetches agent arguments required for a list of skills.

  **Mandatory Parameters:**

  - `skills` (list): A list of skill objects, each containing the `skill_id` and optionally the `skill_version`.

  **Functionality:**
  This method retrieves the agent arguments for a list of skills. It returns the 'properties' and 'required' fields of the skills' input schemas, which describe the arguments needed for the skills to function correctly.

---

**Usage Example:**

```
from skills import Skills

skills = Skills(base_url="https://api.example.com", client_id="your-client-id",
client_secret="your-client-secret", token_url="https://api.example.com/token")
skill_details = await skills.get_skill_by_id(skill_id="skill-uuid", version="1")
search_results = await skills.search_skills(query="search-term")
```

### AgentManager Class

Manages agents via the broker.

**Key Methods:**

- **`get_all_agents(skip=0, limit=50) -> dict`**
  Retrieves all agents with pagination support.

  **Mandatory Parameters:**

  - None

  **Optional Parameters:**

  - `skip` (int): The number of items to skip (for pagination). Default is `0`.
  - `limit` (int): The number of items to return. Default is `50`.

  **Functionality:**
  This method retrieves all agents with pagination support. You can specify how many agents to skip and how many to return in the response.

---

- **`get_agent_by_id(agent_id: str, version: str = '1') -> dict`**
  Retrieves agent details by agent ID.

  **Mandatory Parameters:**

  - `agent_id` (str): The ID of the agent to fetch.

**Optional Parameters:**

- `version` (str): The version of the agent. Default is `'1'`.

**Functionality:**
This method fetches the details of an agent using the `agent_id`. You can optionally specify the `version` of the agent.

---

- **`invoke_agent(agent_id: str, messages: list, **kwargs) -> dict`**
  Invokes an agent with a list of messages.

  **Mandatory Parameters:**

  - `agent_id` (str): The ID of the agent to invoke.
  - `messages` (list): The list of messages to send to the agent.

  **Optional Parameters:**

  - `kwargs`: Additional keyword arguments for the request. These can be used to provide extra parameters or overrides for the invocation.

  **Functionality of `kwargs`:**
  `kwargs` allows passing extra parameters to the request, such as additional configurations or options specific to the agent invocation. These parameters will be added to the request body.

---

- **`create_agent(name: str, description: str, agent_type: str, **kwargs) -> AutoAgent | ControlledAgent | dict`**
  Creates an agent based on the specified type.

  **Mandatory Parameters:**

  - `name` (str): The name of the agent.
  - `description` (str): The description of the agent.
  - `agent_type` (str): The type of agent to create ('auto' or 'controlled').

  **Optional Parameters:**

  - `kwargs`: Additional parameters for initializing the agent.

  **Functionality:**
  This method initializes an agent based on the specified `agent_type`. If the type is `controlled`, it initializes a `ControlledAgent` (refer AgentManager); if it's `auto`, it initializes an `AutoAgent` (refer AgentManager). Any additional parameters for the agent can be passed through `kwargs`. To build the Agent in the backend please follow the AgentManager Docs

---

- **`get_agent_args_by_id(agent_id: str, version: str = 1) -> dict`**
  Retrieves agent arguments by agent ID.

  **Mandatory Parameters:**

  - `agent_id` (str): The ID of the agent.

  **Optional Parameters:**

  - `version` (str): The version of the agent. Default is `'1'`.

**Functionality:**
This method fetches the agent arguments for a specific agent. It checks the agent's skills and retrieves the required arguments based on those skills.

---

- **`create_agent_json(body) -> dict`**
  Creates an agent via a JSON request.

  **Mandatory Parameters:**

  - `body` (dict): The body of the request containing agent information.

  **Functionality:**
  This method creates an agent using the provided JSON `body`, which contains all necessary information about the agent. The agent is created via a POST request to the broker.

---

**Usage Example:**

```
from agent_manager import AgentManager

agent_manager = AgentManager(base_url="https://api.example.com", client_id="your-client-id", client_secret="your-client-secret",
token_url="https://api.example.com/token")
all_agents = await agent_manager.get_all_agents()

# Create an auto agent
auto_agent = agent_manager.create_agent(name="Customer Support Bot",
description="Handles customer queries", agent_type="auto")
auto_agent.add_skill(skill_id="skill-uuid")
await auto_agent.build()

# Create a controlled agent
controlled_agent = agent_manager.create_agent(name="Controlled Agent",
description="Handles complex workflows", agent_type="controlled")
controlled_agent.add_conditional_node(skill_id="skill-uuid", condition={"destination": {"bad": "END", "good": "next-uuid"}, "deciding_fn": {"skill_id": "decide-uuid"}})
await controlled_agent.build()
```

---

## 2. AgentManager Subclasses

### ControlledAgent

Represents a controlled agent with conditional and normal nodes.

**Key Methods:**

- **`add_conditional_node(skill_id: str, condition: Dict[str, Any], skill_version: str = '1', **kwargs) -> ControlledAgent`**
  Adds a conditional node to the agent.

  **Mandatory Parameters:**

- `skill_id` (str): The UUID of the skill to add.
- `condition` (dict): A dictionary representing the condition for the node.

**Optional Parameters:**

- `skill_version` (str): The version of the skill. Default is `'1'`.
- `kwargs`: Additional parameters for the node.

**Functionality:**
This method adds a conditional node to the agent's decision-making process. The node is only added if the condition is valid. Any additional parameters passed in `kwargs` are also added to the node.

---

- **`add_normal_node(skill_id: str, destination: str, skill_version: str = '1', **kwargs) -> ControlledAgent`**
  Adds a normal node to the agent.

  **Mandatory Parameters:**

  - `skill_id` (str): The UUID of the skill to add.
  - `destination` (str): The destination of the node (should be a valid UUID or 'END').

  **Optional Parameters:**

  - `skill_version` (str): The version of the skill. Default is `'1'`.
  - `kwargs`: Additional parameters for the node.

  **Functionality:**
  This method adds a normal node to the agent. The destination must be a valid UUID or `'END'`. Any additional parameters passed in `kwargs` are also added to the node.

---

- **`build() -> dict`**
  Builds the controlled agent by sending its data to the broker.

  **Mandatory Parameters:**

  - None

  **Optional Parameters:**

  - None

  **Functionality:**
  This method builds the controlled agent by sending its data to the broker. It returns the response from the broker after building the agent.

**Usage Example:**

```
controlled_agent = agent_manager.create_agent(name="Controlled Agent",
description="Handles complex workflows", agent_type="controlled")
controlled_agent.add_conditional_node(skill_id="skill-uuid", condition={"destination":
{"bad": "END", "good": "next-uuid"}, "deciding_fn": {"skill_id": "decide-uuid"}})
await controlled_agent.build()
```

### AutoAgent

Represents an auto agent that can add skills and sub-agents.

**Key Methods:**

- **`add_skill(skill_id: str, skill_version=1, **kwargs) -> AutoAgent`**
  Adds a skill to the agent.

  **Mandatory Parameters:**

  - `skill_id` (str): The UUID of the skill to add.

  **Optional Parameters:**

  - `skill_version` (int): The version of the skill. Default is `1`.
  - `kwargs`: Additional parameters for the skill.

  **Functionality:**
  This method adds a skill to the agent. If the skill ID is valid, it appends the skill to the agent's list of skills. Any additional parameters passed in `kwargs` are included with the skill.

---

- **`add_agent(agent_id: str, **kwargs) -> AutoAgent`**
  Adds an agent to this agent.

  **Mandatory Parameters:**

  - `agent_id` (str): The UUID of the agent to add.

  **Optional Parameters:**

  - `kwargs`: Additional parameters for the agent.

  **Functionality:**
  This method adds another agent to the current agent's list. The agent is added if the `agent_id` is valid, and any additional parameters from `kwargs` are included with the agent.

---

- **`build() -> dict`**
  Builds the agent by sending its data to the broker.

  **Mandatory Parameters:**

  - None

  **Optional Parameters:**

  - None

  **Functionality:**
  This method builds the agent by sending its data to the broker and returns the response from the broker.

---

**Usage Example:**

```
auto_agent = agent_manager.create_agent(name="Auto Agent", description="Handles simple workflows", agent_type="auto")
auto_agent.add_skill(skill_id="skill-uuid")
await auto_agent.build()
```

---

## 3. Utilities

### Authentication

Handles authentication for accessing the API, including acquiring and refreshing access tokens.

**Key Methods:**

- **`authenticate() -> str`**: Authenticates the user and retrieves an access token.
- **`refresh_token() -> None`**: Refreshes the access token if it has expired.
- **`get_headers() -> dict`**: Returns the necessary authentication headers for API requests.

**Usage Example:**

```
from authentication import Authentication

auth = Authentication(client_id="your-client-id", client_secret="your-client-secret", base_url="https://api.example.com", token_url="https://api.example.com/token")
headers = await auth.get_headers()
```

### Broker

Handles communication with the remote service via HTTP requests. This class is used internally by other classes.

**Key Methods:**

- **`get(endpoint: str, params: Dict[str, Any] = None, headers: Dict[str, str] = None) -> Dict[str, Any]`**: Sends a GET request to the specified endpoint.
- **`post(endpoint: str, query_params=None, data: Dict[str, Any] = None, headers: Dict[str, str] = None) -> Dict[str, Any]`**: Sends a POST request to the specified endpoint.

**Usage Example:**

```
from broker import Broker

broker = Broker(base_url="https://api.example.com", client_id="your-client-id", client_secret="your-client-secret", token_url="https://api.example.com/token")
response = await broker.get("/skills/search", params={"query": "search-term"})
```

---

This documentation provides a detailed guide to using the Agents-Skill SDK. If you need more examples or details on any function, feel free to ask!