# `OpenAIChat` Documentation

## Table of Contents

## 1. Introduction <a name="introduction"></a>

The `OpenAIChat` class is part of the LangChain library and serves as an interface to interact with OpenAI's Chat large language models. This documentation provides an in-depth understanding of the class, its attributes, methods, and usage examples.

## 2. Class Overview <a name="class-overview"></a>

The `OpenAIChat` class is designed for conducting chat-like conversations with OpenAI's language models, such as GPT-3.5 Turbo. It allows you to create interactive conversations by sending messages and receiving model-generated responses. This class simplifies the process of integrating OpenAI's models into chatbot applications and other natural language processing tasks.

## 3. Class Architecture <a name="class-architecture"></a>

The `OpenAIChat` class is built on top of the `BaseLLM` class, which provides a foundation for working with large language models. This inheritance-based architecture allows for customization and extension while adhering to object-oriented programming principles.

## 4. Class Attributes <a name="class-attributes"></a>

Here are the key attributes and their descriptions for the `OpenAIChat` class:

Attribute	Description
`client`	An internal client for making API calls to OpenAI.
`model_name`	The name of the language model to use (default: "gpt-3.5-turbo").
`model_kwargs`	Additional model parameters valid for `create` calls not explicitly specified.
`openai_api_key`	The OpenAI API key used for authentication.
`openai_api_base`	The base URL for the OpenAI API.
`openai_proxy`	An explicit proxy URL for OpenAI requests.

`max_retries`	The maximum number of retries to make when generating (default: 6).
`prefix_messages`	A list of messages to set the initial conversation state (default: []).
`streaming`	Whether to stream the results or not (default: False).
`allowed_special`	A set of special tokens that are allowed (default: an empty set).
`disallowed_special`	A collection of special tokens that are not allowed (default: "all").

## 5. Methods <a name="methods"></a>

### 5.1 Construction <a name="construction"></a>

#### 5.1.1 `__init__(self, model_name: str = "gpt-3.5-turbo", openai_api_key: Optional[str] = None, openai_api_base: Optional[str] = None, openai_proxy: Optional[str] = None, max_retries: int = 6, prefix_messages: List = [])`

- Description: Initializes an OpenAIChat object.
- Arguments:
  - `model_name` (str): The name of the language model to use (default: "gpt-3.5-turbo").
  - `openai_api_key` (str, optional): The OpenAI API key used for authentication.
  - `openai_api_base` (str, optional): The base URL for the OpenAI API.
  - `openai_proxy` (str, optional): An explicit proxy URL for OpenAI requests.
  - `max_retries` (int): The maximum number of retries to make when generating (default: 6).
  - `prefix_messages` (List): A list of messages to set the initial conversation state (default: []).

### 5.2 Configuration <a name="configuration"></a>

#### 5.2.1 `build_extra(self, values: Dict[str, Any]) -> Dict[str, Any]`

- Description: Builds extra kwargs from additional parameters passed in.

- Arguments:

  - `values` (dict): Values and parameters to build extra kwargs.

- Returns:

  - Dict[str, Any]: A dictionary of built extra kwargs.

#### 5.2.2 `validate_environment(self, values: Dict) -> Dict`

- Description: Validates that the API key and Python package exist in the environment.

- Arguments:

  - `values` (dict): The class values and parameters.

- Returns:

  - Dict: A dictionary of validated values.

### 5.3 Message Handling <a name="message-handling"></a>

#### 5.3.1 `_get_chat_params(self, prompts: List[str], stop: Optional[List[str]] = None) -> Tuple`

- Description: Gets chat-related parameters for generating responses.

- Arguments:

  - `prompts` (list): List of user messages.

  - `stop` (list, optional): List of stop words.

- Returns:

  - Tuple: Messages and parameters.

### 5.4 Generation <a name="generation"></a>

#### 5.4.1 `_stream(self, prompt: str, stop: Optional[List[str]] = None, run_manager:

Optional[CallbackManagerForLLMRun] = None, **kwargs: Any) -> Iterator[GenerationChunk]`

- Description: Generates text asynchronously using the OpenAI API.

- Arguments:

  - `prompt` (str): The user's message.

  - `stop` (list, optional): List of stop words.

  - `run_manager` (optional): Callback manager for asynchronous generation.

  - `**kwargs` (dict): Additional parameters for asynchronous generation.

- Returns:

  - Iterator[GenerationChunk]: An iterator of generated text chunks.

#### 5.4.2 `_agenerate(self, prompts: List[str], stop: Optional[List[str]] = None, run_manager: Optional[AsyncCallbackManagerForLLMRun] = None, **kwargs: Any) -> LLMResult`

- Description: Generates text asynchronously using the OpenAI API (async version).

- Arguments:

  - `prompts` (list): List of user messages.

  - `stop` (list, optional): List of stop words.

  - `run_manager` (optional): Callback manager for asynchronous generation.

  - `**kwargs` (dict): Additional parameters for asynchronous generation.

- Returns:

  - LLMResult: A result object containing the generated text.

### 5.5 Tokenization <a name="tokenization"></a>

#### 5.5.1 `get_token_ids(self, text: str) -> List[int]`

- Description: Gets token IDs using the tiktoken package.

- Arguments:

  - `text` (str): The text for which to calculate token IDs.

- Returns:

  - List[int]: A list of

 token IDs.

## 6. Usage Examples <a name="usage-examples"></a>

### Example 1: Initializing `OpenAIChat`

```python
from swarm_models import OpenAIChat

# Initialize OpenAIChat with model name and API key
openai_chat = OpenAIChat(model_name="gpt-3.5-turbo", openai_api_key="YOUR_API_KEY")
```

### Example 2: Sending Messages and Generating Responses

```python
# Define a conversation
conversation = [
    "User: Tell me a joke.",
    "Assistant: Why did the chicken cross the road?",
    "User: I don't know. Why?",
    "Assistant: To get to the other side!",
]

# Set the conversation as the prefix messages
openai_chat.prefix_messages = conversation

# Generate a response
user_message = "User: Tell me another joke."
response = openai_chat.generate([user_message])

# Print the generated response
print(
    response[0][0].text
)  # Output: "Assistant: Why don't scientists trust atoms? Because they make up everything!"
```

### Example 3: Asynchronous Generation

```python
import asyncio


# Define an asynchronous function for generating responses
async def generate_responses():
    user_message = "User: Tell me a fun fact."
    async for chunk in openai_chat.stream([user_message]):
        print(chunk.text)
```

```
# Run the asynchronous generation function
asyncio.run(generate_responses())
```

## 7. Additional Information <a name="additional-information"></a>

- To use the `OpenAIChat` class, you should have the `openai` Python package installed, and the environment variable `OPENAI_API_KEY` set with your API key.
- Any parameters that are valid to be passed to the `openai.create` call can be passed to the `OpenAIChat` constructor.
- You can customize the behavior of the class by setting various attributes, such as `model_name`, `openai_api_key`, `prefix_messages`, and more.
- For asynchronous generation, you can use the `_stream` and `_agenerate` methods to interactively receive model-generated text chunks.
- To calculate token IDs, you can use the `get_token_ids` method, which utilizes the `tiktoken` package. Make sure to install the `tiktoken` package with `pip install tiktoken` if needed.

This documentation provides a comprehensive overview of the `OpenAIChat` class, its attributes, methods, and usage examples. You can use this class to create chatbot applications, conduct conversations with language models, and explore the capabilities of OpenAI's GPT-3.5 Turbo model.