

semantic_kernel Package

Reference

Packages

[] Expand table

connectors
contents
exceptions
functions
prompt_template
services
text

Modules

[] Expand table

const
kernel
kernel_pydantic

Classes

[] Expand table

Kernel The Kernel class is the main entry point for the Semantic Kernel. It provides the ability to run semantic/native functions, and manage plugins, memory, and AI services.

Initialize a new instance of the Kernel class.

connectors Package

Reference

Packages

[] [Expand table](#)

ai
memory
openai_plugin
openapi_plugin
search_engine
utils

Modules

[] [Expand table](#)

telemetry

ai Package

Reference

Packages

[] Expand table

open_ai

Modules

[] Expand table

chat_completion_client_base
function_call_behavior
prompt_execution_settings
text_completion_client_base

Classes

[] Expand table

PromptExecutionSettings	Base class for prompt execution settings.
---	---

Can be used by itself or as a base class for other prompt execution settings. The methods are used to create specific prompt execution settings objects based on the keys in the extension_data field, this way you can create a generic PromptExecutionSettings object in your application, which get's mapped into the keys of the prompt execution settings that each services returns by using the service.get_prompt_execution_settings() method.

open_ai Package

Reference

Modules

[] [Expand table](#)

<code>const</code>

Classes

[] [Expand table](#)

ApiKeyAuthentication	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
AzureAIDataSource	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
AzureAIDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
AzureChatCompletion	Azure Chat completion class.

	Initialize an AzureChatCompletion service.
AzureChatPromptExecutionSettings	Specific settings for the Azure OpenAI Chat Completion endpoint.
AzureCosmosDBDataSource	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><i>Raises [ValidationError][pydantic_core.ValidationError]</i> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureCosmosDBDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><i>Raises [ValidationError][pydantic_core.ValidationError]</i> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><i>Raises [ValidationError][pydantic_core.ValidationError]</i> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureEmbeddingDependency	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><i>Raises [ValidationError][pydantic_core.ValidationError]</i> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureTextCompletion	<p>Azure Text Completion class.</p> <p>Initialize an AzureTextCompletion service.</p>
AzureTextEmbedding	<p>Azure Text Embedding class.</p> <p>Note: This class is experimental and may change in the future.</p>

Initialize an AzureTextEmbedding service.

service_id: The service ID. (Optional) api_key {str | None}: The optional api key. If provided, will override the value in the

env vars or .env file.

deployment_name {str | None}: The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file.

endpoint {str | None}: The optional deployment endpoint. If provided will override the value in the env vars or .env file.

base_url {str | None}: The optional deployment base_url. If provided will override the value in the env vars or .env file.

api_version {str | None}: The optional deployment api version. If provided will override the value in the env vars or .env file.

ad_token {str | None}: The Azure AD token for authentication. (Optional) ad_auth {AsyncAzureADTokenProvider | None}: Whether to use Azure Active Directory authentication.

(Optional) The default value is False.

default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

async_client {Optional[AsyncAzureOpenAI]} – An existing client to use. (Optional) env_file_path {str | None} – Use the environment settings file as a fallback to

environment variables. (Optional)

ConnectionStringAuthentication

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

DataSourceFieldsMapping

Create a new model by parsing and validating input data from keyword arguments.

	<p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
ExtraBody	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
OpenAIChatCompletion	<p>OpenAI Chat completion class.</p> <p>Initialize an OpenAIChatCompletion service.</p> <p>:param : the env vars or .env file value. :param org_id {str None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)</p>
OpenAIChatPromptExecutionSettings	Specific settings for the Chat Completion endpoint.
OpenAIPromptExecutionSettings	Common request settings for (Azure) OpenAI services.
OpenAITextCompletion	<p>OpenAI Text Completion class.</p> <p>Initialize an OpenAITextCompletion service.</p> <p>:param : the env vars or .env file value. :param org_id {str None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)</p>
OpenAITextEmbedding	<p>OpenAI Text Embedding class.</p> <p>Note: This class is experimental and may change in the future.</p> <p>Initializes a new instance of the OpenAITextCompletion class.</p>

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers {Optional[Mapping[str: The default headers mapping of string keys to string values for HTTP requests. (Optional)

OpenAITextPromptExecutionSettings	Specific settings for the completions endpoint.
---	---

const Module

Reference

content_filter_ai_exception Module

Reference

Classes

[+] Expand table

ContentFilterAIException	AI exception for an error from Azure OpenAI's content filter
	Initializes a new instance of the ContentFilterAIException class.
ContentFilterResult	

Enums

[+] Expand table

ContentFilterCodes
ContentFilterResultSeverity

ContentFilterAIException Class

Reference

AI exception for an error from Azure OpenAI's content filter

Initializes a new instance of the ContentFilterAIException class.

Inheritance [ServiceContentFilterException](#) → ContentFilterAIException

Constructor

Python

```
ContentFilterAIException(message: str, inner_exception: BadRequestError)
```

Parameters

[+] [Expand table](#)

Name	Description
message. Required*	<xref:<xref:message {str} -- The error>>
exception. Required*	<xref:<xref:inner_exception {Exception} -- The inner>>
message Required*	
inner_exception Required*	

Attributes

content_filter_code

Python

```
content_filter_code: ContentFilterCodes
```

content_filter_result

Python

```
content_filter_result: dict[str,  
semantic_kernel.connectors.ai.open_ai.exceptions.content_filter_ai_except  
ion.ContentFilterResult]
```

param

Python

```
param: str
```

ContentFilterCodes Enum

Reference

Inheritance [Enum](#) → ContentFilterCodes

Constructor

Python

```
ContentFilterCodes(value, names=None, *, module=None, qualname=None,  
type=None, start=1, boundary=None)
```

Fields

[\[+\]](#) [Expand table](#)

RESPONSIBLE_AI_POLICY_VIOLATION

ContentFilterResult Class

Reference

Inheritance builtins.object → ContentFilterResult

Constructor

Python

```
ContentFilterResult(filtered: bool = False, detected: bool = False,  
severity:  
    semantic_kernel.connectors.ai.open_ai.exceptions.content_filter_ai_exception  
.ContentFilterResultSeverity = <ContentFilterResultSeverity.SAFE: ''safe''>)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>filtered</code>	default value: False
<code>detected</code>	default value: False
<code>severity</code>	default value: ContentFilterResultSeverity.SAFE

Methods

[\[\] Expand table](#)

<code>from_inner_error_result</code>	Creates a ContentFilterResult from the inner error results.
--------------------------------------	---

`from_inner_error_result`

Creates a ContentFilterResult from the inner error results.

Python

```
from_inner_error_result(inner_error_results: dict[str, Any]) ->  
ContentFilterResult
```

Parameters

[\[\] Expand table](#)

Name	Description
from. Required*	<xref:<xref:key {str} -- The key to get the inner error result>>
{Dict[str] Required*	str (<xref:inner_error_results>
results. Required*	Any]<xref:{} -- The inner error>
inner_error_results Required*	

Returns

[\[\] Expand table](#)

Type	Description
	ContentFilterResult – The ContentFilterResult.

Attributes

detected

```
Python  
  
detected: bool = False
```

filtered

```
Python  
  
filtered: bool = False
```

severity

Python

```
severity: ContentFilterResultSeverity = 'safe'
```

ContentFilterResultSeverity Enum

Reference

Inheritance [Enum](#) → ContentFilterResultSeverity

Constructor

Python

```
ContentFilterResultSeverity(value, names=None, *, module=None,  
qualname=None, type=None, start=1, boundary=None)
```

Fields

[+] [Expand table](#)

HIGH
MEDIUM
SAFE

azure_chat_prompt_execution_settings

Module

Reference

Classes

[] Expand table

ApiKeyAuthentication	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureAIDataSource	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureAIDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureChatPromptExecutionSettings	Specific settings for the Azure OpenAI Chat Completion endpoint.
AzureChatRequestBase	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p>

	<p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureCosmosDBDataSource	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureCosmosDBDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureDataSourceParameters	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
AzureEmbeddingDependency	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
ConnectionStringAuthentication	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>

[DataSourceFieldsMapping](#)

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

[ExtraBody](#)

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

ApiKeyAuthentication Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [AzureChatRequestBase](#) → ApiKeyAuthentication

Constructor

Python

```
ApiKeyAuthentication(*, type: Literal['APIKey', 'api_key'] = 'api_key', key: str | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>type</code>	default value: <code>api_key</code>
<code>key</code> Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'key':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='key', serialization_alias='key'),  
    'type': FieldInfo(annotation=Literal['APIKey', 'api_key'],  
    required=False, default='api_key', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

key

Python

```
key: str | None
```

type

Python

```
type: Literal['APIKey', 'api_key']
```

AzureAIDataSource Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureAIDataSource

Constructor

Python

```
AzureAIDataSource(*, type: Literal['azure_search'] = 'azure_search',  
parameters: dict, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
type	default value: azure_search
parameters Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'parameters':  
    FieldInfo(annotation=dict, required=True, alias_priority=1,  
    validation_alias='parameters', serialization_alias='parameters',  
    metadata=[<class  
        'semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_ch  
        at_prompt_execution_settings.AzureAIDataSourceParameters'>]),  
    'type': FieldInfo(annotation=Literal['azure_search'], required=False,  
    default='azure_search', alias_priority=1, validation_alias='type',  
    serialization_alias='type')}
```

parameters

Python

```
parameters: dict
```

type

Python

```
type: Literal['azure_search']
```

AzureAIDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureDataSourceParameters](#) → AzureAIDataSourceParameters

Constructor

Python

```
AzureAIDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, endpoint: str | None = None, queryType: Literal['simple', 'semantic', 'vector', 'vectorSimpleHybrid', 'vectorSemanticHybrid'] = 'simple', authentication: ApiKeyAuthentication | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True

Name	Description
topNDocuments	default value: 5
semanticConfiguration Required*	
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	
endpoint Required*	
queryType	default value: simple
authentication Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,
```

```
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'authentication':  
        FieldInfo(annotation=Union[ApiKeyAuthentication, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='authentication', serialization_alias='authentication'),  
    'embedding_dependency':  
        FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='embeddingDependency',  
                  serialization_alias='embedding_dependency'),  
    'endpoint':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='endpoint',  
                  serialization_alias='endpoint'),  
    'fields_mapping':  
        FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
    'filter':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='filter',  
                  serialization_alias='filter'),  
    'in_scope':  
        FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
                  alias_priority=1, validation_alias='inScope',  
                  serialization_alias='in_scope'),  
    'index_language':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='indexLanguage',  
                  serialization_alias='index_language'),  
    'index_name':  
        FieldInfo(annotation=str, required=True, alias_priority=1,  
                  validation_alias='indexName', serialization_alias='index_name'),  
    'query_type':  
        FieldInfo(annotation=Literal['simple', 'semantic',  
                                    'vector', 'vectorSimpleHybrid', 'vectorSemanticHybrid'], required=False,  
                  default='simple', alias_priority=1, validation_alias='queryType',  
                  serialization_alias='query_type', metadata=[AfterValidator(func=<function  
to_snake>)]),  
    'role_information':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None, alias_priority=1,  
                  validation_alias='roleInformation',  
                  serialization_alias='role_information'),  
    'semantic_configuration':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='semanticConfiguration',  
                  serialization_alias='semantic_configuration'),  
    'strictness':  
        FieldInfo(annotation=int, required=False, default=3, alias_priority=1,
```

```
validation_alias='strictness', serialization_alias='strictness'),
'top_n_documents': FieldInfo(annotation=Union[int, NoneType],
required=False, default=5, alias_priority=1,
validation_alias='topNDocuments', serialization_alias='top_n_documents'})
```

authentication

Python

```
authentication: ApiKeyAuthentication | None
```

embedding_dependency

Python

```
embedding_dependency: AzureEmbeddingDependency | None
```

endpoint

Python

```
endpoint: str | None
```

fields_mapping

Python

```
fields_mapping: DataSourceFieldsMapping | None
```

filter

Python

```
filter: str | None
```

in_scope

Python

```
in_scope: bool | None
```

index_language

```
Python
```

```
index_language: str | None
```

index_name

```
Python
```

```
index_name: str
```

query_type

```
Python
```

```
query_type: Literal['simple', 'semantic', 'vector', 'vectorSimpleHybrid',  
'vectorSemanticHybrid']
```

role_information

```
Python
```

```
role_information: str | None
```

semantic_configuration

```
Python
```

```
semantic_configuration: str | None
```

strictness

```
Python
```

```
strictness: int
```

top_n_documents

Python

```
top_n_documents: int | None
```

AzureChatPromptExecutionSettings Class

Reference

Specific settings for the Azure OpenAI Chat Completion endpoint.

Inheritance [OpenAIChatPromptExecutionSettings](#) →
AzureChatPromptExecutionSettings

Constructor

Python

```
AzureChatPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
response_format: str | None = None, tools: list[dict[str, Any]] | None =  
None, tool_choice: str | None = None, function_call: str | None = None,  
functions: list[dict[str, Any]] | None = None, messages: list[dict[str,  
Any]] | None = None, function_call_behavior: FunctionCallBehavior | None =  
None, extra_body: dict[str, Any] | ExtraBody | None = None)
```

Parameters

[+] Expand table

Name	Description
<code>service_id</code>	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>extension_data</code>	

Name	Description
Required*	
ai_model_id	
Required*	
frequency_penalty	
Required*	
logit_bias	
Required*	
max_tokens	
Required*	
number_of_responses	
Required*	
presence_penalty	
Required*	
seed	
Required*	
stop	
Required*	
stream	
Required*	
temperature	
Required*	
top_p	
Required*	
user	
Required*	
response_format	
Required*	
tools	
Required*	
tool_choice	
Required*	
function_call	
Required*	
functions	

Name	Description
Required*	
messages	
Required*	
function_call_behavior	
Required*	
extra_body	
Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=2, serialization_alias='model'), 'extension_data':  
    FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'extra_body': FieldInfo(annotation=Union[dict[str,  
    Any], ExtraBody, NoneType], required=False, default=None),  
    'frequency_penalty': FieldInfo(annotation=Union[float, NoneType],  
    required=False, default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]),  
    'function_call': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'function_call_behavior':  
    FieldInfo(annotation=Union[FunctionCallBehavior, NoneType],  
    required=False, default=None, exclude=True), 'functions':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None), 'logit_bias':  
    FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],  
    required=False, default=None), 'max_tokens':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
    metadata=[Gt(gt=0)]), 'messages':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None), 'number_of_responses':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
    alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),  
    Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,  
    NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),  
    Le(le=2.0)]), 'response_format': FieldInfo(annotation=Union[str,  
    NoneType], required=False, default=None), 'seed':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None),  
    'service_id': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None, metadata=[MinLen(min_length=1)]), 'stop':  
    FieldInfo(annotation=Union[str, list[str], NoneType], required=False,  
    default=None), 'stream': FieldInfo(annotation=bool, required=False,  
    default=False), 'temperature': FieldInfo(annotation=Union[float,  
    NoneType], required=False, default=None, metadata=[Ge(ge=0.0),  
    Le(le=2.0)]), 'tool_choice': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'tools':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None, metadata=[MaxLen(max_length=64)]), 'top_p':  
    FieldInfo(annotation=Union[float, NoneType], required=False,  
    default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

extension_data

Python

```
extension_data: dict[str, Any]
```

extra_body

Python

```
extra_body: dict[str, Any] | ExtraBody | None
```

frequency_penalty

Python

```
frequency_penalty: float | None
```

function_call

Python

```
function_call: str | None
```

function_call_behavior

Python

```
function_call_behavior: FunctionCallBehavior | None
```

functions

Python

```
functions: list[dict[str, Any]] | None
```

logit_bias

Python

```
logit_bias: dict[str | int, float] | None
```

max_tokens

Python

```
max_tokens: int | None
```

messages

Python

```
messages: list[dict[str, Any]] | None
```

number_of_responses

Python

```
number_of_responses: int | None
```

presence_penalty

Python

```
presence_penalty: float | None
```

response_format

Python

```
response_format: str | None
```

seed

Python

```
seed: int | None
```

service_id

Python

```
service_id: str | None
```

stop

Python

```
stop: str | list[str] | None
```

stream

Python

```
stream: bool
```

temperature

Python

```
temperature: float | None
```

tool_choice

Python

```
tool_choice: str | None
```

tools

Python

```
tools: list[dict[str, Any]] | None
```

top_p

Python

```
top_p: float | None
```

user

Python

```
user: str | None
```

AzureChatRequestBase Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → AzureChatRequestBase

Constructor

Python

```
AzureChatRequestBase(**extra_data: Any)
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':
```

```
True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {}
```

AzureCosmosDBDataSource Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureCosmosDBDataSource

Constructor

Python

```
AzureCosmosDBDataSource(*, type: Literal['azure_cosmos_db'] =  
    'azure_cosmos_db', parameters: AzureCosmosDBDataSourceParameters,  
    **extra_data: Any)
```

Keyword-Only Parameters

[] [Expand table](#)

Name	Description
type	default value: azure_cosmos_db
parameters Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'parameters':  
    FieldInfo(annotation=AzureCosmosDBDataSourceParameters, required=True,  
    alias_priority=1, validation_alias='parameters',  
    serialization_alias='parameters'), 'type':  
    FieldInfo(annotation=Literal['azure_cosmos_db'], required=False,  
    default='azure_cosmos_db', alias_priority=1, validation_alias='type',  
    serialization_alias='type')}
```

parameters

Python

```
parameters: AzureCosmosDBDataSourceParameters
```

type

Python

```
type: Literal['azure_cosmos_db']
```

AzureCosmosDBDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureDataSourceParameters](#) → AzureCosmosDBDataSourceParameters

Constructor

Python

```
AzureCosmosDBDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, authentication: ConnectionStringAuthentication | None = None, databaseName: str | None = None, containerName: str | None = None, embeddingDependencyType: AzureEmbeddingDependency | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True

Name	Description
topNDocuments	default value: 5
semanticConfiguration Required*	
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	
authentication Required*	
databaseName Required*	
containerName Required*	
embeddingDependencyType Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'authentication':  
    FieldInfo(annotation=Union[ConnectionStringAuthentication, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='authentication', serialization_alias='authentication'),  
    'container_name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='containerName', serialization_alias='container_name'),  
    'database_name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='databaseName', serialization_alias='database_name'),  
    'embedding_dependency':  
        FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
        required=False, default=None, alias_priority=1,  
        validation_alias='embeddingDependency',  
        serialization_alias='embedding_dependency'), 'embedding_dependency_type':  
            FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
            required=False, default=None, alias_priority=1,  
            validation_alias='embeddingDependencyType',  
            serialization_alias='embedding_dependency_type'), 'fields_mapping':  
                FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
                required=False, default=None, alias_priority=1,  
                validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
                'filter': FieldInfo(annotation=Union[str, NoneType], required=False,  
                default=None, alias_priority=1, validation_alias='filter',  
                serialization_alias='filter'), 'in_scope':  
                    FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
                    alias_priority=1, validation_alias='inScope',  
                    serialization_alias='in_scope'), 'index_language':  
                        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                        alias_priority=1, validation_alias='indexLanguage',  
                        serialization_alias='index_language'), 'index_name':  
                            FieldInfo(annotation=str, required=True, alias_priority=1,  
                            validation_alias='indexName', serialization_alias='index_name'),  
                            'role_information': FieldInfo(annotation=Union[str, NoneType],
```

```
required=False, default=None, alias_priority=1,
validation_alias='roleInformation',
serialization_alias='role_information'), 'semantic_configuration':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=1, validation_alias='semanticConfiguration',
serialization_alias='semantic_configuration'), 'strictness':
FieldInfo(annotation=int, required=False, default=3, alias_priority=1,
validation_alias='strictness', serialization_alias='strictness'),
'top_n_documents': FieldInfo(annotation=Union[int, NoneType],
required=False, default=5, alias_priority=1,
validation_alias='topNDocuments', serialization_alias='top_n_documents')})
```

authentication

Python

```
authentication: ConnectionStringAuthentication | None
```

container_name

Python

```
container_name: str | None
```

database_name

Python

```
database_name: str | None
```

embedding_dependency

Python

```
embedding_dependency: AzureEmbeddingDependency | None
```

embedding_dependency_type

Python

```
embedding_dependency_type: AzureEmbeddingDependency | None
```

fields_mapping

Python

```
fields_mapping: DataSourceFieldsMapping | None
```

filter

Python

```
filter: str | None
```

in_scope

Python

```
in_scope: bool | None
```

index_language

Python

```
index_language: str | None
```

index_name

Python

```
index_name: str
```

role_information

Python

```
role_information: str | None
```

semantic_configuration

Python

```
semantic_configuration: str | None
```

strictness

Python

```
strictness: int
```

top_n_documents

Python

```
top_n_documents: int | None
```

AzureDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureDataSourceParameters

Constructor

Python

```
AzureDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True
topNDocuments	default value: 5
semanticConfiguration Required*	

Name	Description
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator': AliasGenerator(alias=None, validation_alias=<function to_camel>, serialization_alias=<function to_snake>), 'arbitrary_types_allowed': True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'embedding_dependency':  
    FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='embeddingDependency',  
    serialization_alias='embedding_dependency'), 'fields_mapping':  
    FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
    'filter': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None, alias_priority=1, validation_alias='filter',  
    serialization_alias='filter'), 'in_scope':  
    FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
    alias_priority=1, validation_alias='inScope',  
    serialization_alias='in_scope'), 'index_language':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='indexLanguage',  
    serialization_alias='index_language'), 'index_name':  
    FieldInfo(annotation=str, required=True, alias_priority=1,  
    validation_alias='indexName', serialization_alias='index_name'),  
    'role_information': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='roleInformation',  
    serialization_alias='role_information'), 'semantic_configuration':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='semanticConfiguration',  
    serialization_alias='semantic_configuration'), 'strictness':  
    FieldInfo(annotation=int, required=False, default=3, alias_priority=1,  
    validation_alias='strictness', serialization_alias='strictness'),  
    'top_n_documents': FieldInfo(annotation=Union[int, NoneType],  
    required=False, default=5, alias_priority=1,  
    validation_alias='topNDocuments', serialization_alias='top_n_documents')})
```

embedding_dependency

Python

```
embedding_dependency: AzureEmbeddingDependency | None
```

fields_mapping

Python

```
fields_mapping: DataSourceFieldsMapping | None
```

filter

Python

```
filter: str | None
```

in_scope

Python

```
in_scope: bool | None
```

index_language

Python

```
index_language: str | None
```

index_name

Python

```
index_name: str
```

role_information

Python

```
role_information: str | None
```

semantic_configuration

Python

```
semantic_configuration: str | None
```

strictness

Python

```
strictness: int
```

top_n_documents

Python

```
top_n_documents: int | None
```

AzureEmbeddingDependency Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureEmbeddingDependency

Constructor

Python

```
AzureEmbeddingDependency(*, type: Literal['DeploymentName',  
    'deployment_name'] = 'deployment_name', deploymentName: str | None = None,  
    **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
type	default value: deployment_name
deploymentName Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'deployment_name':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='deploymentName',  
    serialization_alias='deployment_name'), 'type':  
    FieldInfo(annotation=Literal['DeploymentName', 'deployment_name'],  
    required=False, default='deployment_name', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

deployment_name

Python

```
deployment_name: str | None
```

type

Python

```
type: Literal['DeploymentName', 'deployment_name']
```

ConnectionStringAuthentication Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [AzureChatRequestBase](#) → `ConnectionStringAuthentication`

Constructor

Python

```
ConnectionStringAuthentication(*, type: Literal['ConnectionString',  
'connection_string'] = 'connection_string', connectionString: str | None =  
None, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>type</code>	default value: <code>connection_string</code>
<code>connectionString</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'connection_string':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='connectionString',  
    serialization_alias='connection_string'), 'type':  
    FieldInfo(annotation=Literal['ConnectionString', 'connection_string'],  
    required=False, default='connection_string', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

connection_string

Python

```
connection_string: str | None
```

type

Python

```
type: Literal['ConnectionString', 'connection_string']
```

DataSourceFieldsMapping Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → DataSourceFieldsMapping

Constructor

Python

```
DataSourceFieldsMapping(*, titleField: str | None = None, urlField: str |  
None = None, filepathField: str | None = None, contentFields: list[str] |  
None = None, vectorFields: list[str] | None = None, contentFieldsSeparator:  
str | None = '\n', **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
titleField Required*	
urlField Required*	
filepathField Required*	
contentFields Required*	
vectorFields Required*	
contentFieldsSeparator	default value:

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content_fields':  
    FieldInfo(annotation=Union[list[str], NoneType], required=False,  
    default=None, alias_priority=1, validation_alias='contentFields',  
    serialization_alias='content_fields'), 'content_fields_separator':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default='\n',  
    alias_priority=1, validation_alias='contentFieldsSeparator',  
    serialization_alias='content_fields_separator'), 'filepath_field':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='filepathField',  
    serialization_alias='filepath_field'), 'title_field':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='titleField',
```

```
    serialization_alias='title_field'), 'url_field':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        alias_priority=1, validation_alias='urlField',  
        serialization_alias='url_field'), 'vector_fields':  
        FieldInfo(annotation=Union[list[str], NoneType], required=False,  
        default=None, alias_priority=1, validation_alias='vectorFields',  
        serialization_alias='vector_fields'))}
```

content_fields

Python

```
content_fields: list[str] | None
```

content_fields_separator

Python

```
content_fields_separator: str | None
```

filepath_field

Python

```
filepath_field: str | None
```

title_field

Python

```
title_field: str | None
```

url_field

Python

```
url_field: str | None
```

vector_fields

Python

```
vector_fields: list[str] | None
```

ExtraBody Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → ExtraBody

Constructor

Python

```
ExtraBody(*, data_sources:  
list[typing.Annotated[typing.Union[semantic_kernel.connectors.ai.open_ai.pro  
mpt_execution_settings.azure_chat_prompt_execution_settings.AzureAISearc  
hData  
sSource,  
semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_chat_p  
rompt_execution_settings.AzureCosmosDBDataSource],  
FieldInfo(annotation=NoneType, required=True, discriminator='type')]] | None  
= None, input_language: str | None = None, output_language: str | None =  
None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
data_sources Required*	
input_language Required*	
output_language Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'data_sources':
FieldInfo(annotation=Union[list[Annotated[Union[AzureAIDataSource,
AzureCosmosDBDataSource], FieldInfo(annotation=NoneType, required=True,
discriminator='type')]], NoneType], required=False, default=None),
'input_language': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None, alias_priority=2,
serialization_alias='inputLanguage'), 'output_language':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='outputLanguage')}
```

data_sources

Python

```
data_sources:
list[typing.Annotated[typing.Union[semantic_kernel.connectors.ai.open_ai.
```

```
prompt_execution_settings.azure_chat_prompt_execution_settings.AzureAISe
rchDataSource,
semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_cha
t_prompt_execution_settings.AzureCosmosDBDataSource],
FieldInfo(annotation=NoneType, required=True, discriminator='type')]] | |
None
```

input_language

Python

```
input_language: str | None
```

output_language

Python

```
output_language: str | None
```

open_ai_prompt_execution_settings Module

Reference

Classes

[] [Expand table](#)

OpenAIChatPromptExecutionSettings	Specific settings for the Chat Completion endpoint.
OpenAIEmbeddingPromptExecutionSettings	
OpenAIPromptExecutionSettings	Common request settings for (Azure) OpenAI services.
OpenAITextPromptExecutionSettings	Specific settings for the completions endpoint.

OpenAIChatPromptExecutionSettings Class

Reference

Specific settings for the Chat Completion endpoint.

Inheritance [OpenAIPromptExecutionSettings](#) → OpenAIChatPromptExecutionSettings

Constructor

Python

```
OpenAIChatPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
response_format: dict[Literal['type'], Literal['text', 'json_object']] |  
None = None, tools: list[dict[str, Any]] | None = None, tool_choice: str |  
None = None, function_call: str | None = None, functions: list[dict[str,  
Any]] | None = None, messages: list[dict[str, Any]] | None = None,  
function_call_behavior: FunctionCallBehavior | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
extension_data Required*	

Name	Description
ai_model_id Required*	
frequency_penalty Required*	
logit_bias Required*	
max_tokens Required*	
number_of_responses Required*	
presence_penalty Required*	
seed Required*	
stop Required*	
stream Required*	
temperature Required*	
top_p Required*	
user Required*	
response_format Required*	
tools Required*	
tool_choice Required*	
function_call Required*	
functions Required*	

Name	Description
messages Required*	
function_call_behavior Required*	

Methods

[+] [Expand table](#)

[validate_function_call](#)

validate_function_call

Python

```
validate_function_call(v: str | list[dict[str, Any]] | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
v	default value: None

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='model'), 'extension_data':
FieldInfo(annotation=dict[str, Any], required=False,
default_factory=dict), 'frequency_penalty':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]), 'function_call':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'function_call_behavior':
FieldInfo(annotation=Union[FunctionCallBehavior, NoneType],
required=False, default=None, exclude=True), 'functions':
FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],
required=False, default=None), 'logit_bias':
FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],
required=False, default=None), 'max_tokens':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
metadata=[Gt(gt=0)]), 'messages':
FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],
required=False, default=None), 'number_of_responses':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),
Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,
NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),
Le(le=2.0)]), 'response_format':
FieldInfo(annotation=Union[dict[Literal['type'], Literal['text',
'json_object']], NoneType], required=False, default=None), 'seed':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None),
'service_id': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None, metadata=[MinLen(min_length=1)]), 'stop':
FieldInfo(annotation=Union[str, list[str], NoneType], required=False,
default=None), 'stream': FieldInfo(annotation=bool, required=False)}
```

```
default=False), 'temperature': FieldInfo(annotation=Union[float,  
NoneType], required=False, default=None, metadata=[Ge(ge=0.0),  
Le(le=2.0)]), 'tool_choice': FieldInfo(annotation=Union[str, NoneType],  
required=False, default=None), 'tools':  
FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
required=False, default=None, metadata=[MaxLen(max_length=64)]), 'top_p':  
FieldInfo(annotation=Union[float, NoneType], required=False,  
default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

extension_data

Python

```
extension_data: dict[str, Any]
```

frequency_penalty

Python

```
frequency_penalty: float | None
```

function_call

Python

```
function_call: str | None
```

function_call_behavior

Python

```
function_call_behavior: FunctionCallBehavior | None
```

functions

Python

```
functions: list[dict[str, Any]] | None
```

logit_bias

Python

```
logit_bias: dict[str | int, float] | None
```

max_tokens

Python

```
max_tokens: int | None
```

messages

Python

```
messages: list[dict[str, Any]] | None
```

number_of_responses

Python

```
number_of_responses: int | None
```

presence_penalty

Python

```
presence_penalty: float | None
```

response_format

Python

```
response_format: dict[Literal['type'], Literal['text', 'json_object']] | None
```

seed

Python

```
seed: int | None
```

service_id

Python

```
service_id: str | None
```

stop

Python

```
stop: str | list[str] | None
```

stream

Python

```
stream: bool
```

temperature

Python

```
temperature: float | None
```

tool_choice

Python

```
tool_choice: str | None
```

tools

```
Python
```

```
tools: list[dict[str, Any]] | None
```

top_p

```
Python
```

```
top_p: float | None
```

user

```
Python
```

```
user: str | None
```

OpenAIEmbeddingPromptExecutionSettings Class

Reference

Inheritance [PromptExecutionSettings](#) → OpenAIEmbeddingPromptExecutionSettings

Constructor

Python

```
OpenAIEmbeddingPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, input: str | list[str] | list[int] |  
list[list[int]] | None = None, ai_model_id: str | None = None,  
encoding_format: Literal['float', 'base64'] | None = None, user: str | None  
= None, extra_headers: dict | None = None, extra_query: dict | None = None,  
extra_body: dict | None = None, timeout: float | None = None, dimensions:  
int | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
extension_data	
Required*	
input	
Required*	
ai_model_id	
Required*	
encoding_format	

Name	Description
Required*	
user	
Required*	
extra_headers	
Required*	
extra_query	
Required*	
extra_body	
Required*	
timeout	
Required*	
dimensions	
Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo][pydantic.fields.FieldInfo]`.

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'ai_model_id': FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=2, serialization_alias='model'), 'dimensions':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
    metadata=[Gt(gt=0), Le(le=3072)]), 'encoding_format':  
    FieldInfo(annotation=Union[Literal['float', 'base64'], NoneType],  
    required=False, default=None), 'extension_data':  
    FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'extra_body': FieldInfo(annotation=Union[dict,  
    NoneType], required=False, default=None), 'extra_headers':  
    FieldInfo(annotation=Union[dict, NoneType], required=False,  
    default=None), 'extra_query': FieldInfo(annotation=Union[dict, NoneType],  
    required=False, default=None), 'input': FieldInfo(annotation=Union[str,  
    list[str], list[int], list[list[int]]], NoneType], required=False,  
    default=None), 'service_id': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, metadata=[MinLen(min_length=1)]),  
    'timeout': FieldInfo(annotation=Union[float, NoneType], required=False,  
    default=None), 'user': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

dimensions

Python

```
dimensions: int | None
```

encoding_format

Python

```
encoding_format: Literal['float', 'base64'] | None
```

extra_body

Python

```
extra_body: dict | None
```

extra_headers

Python

```
extra_headers: dict | None
```

extra_query

Python

```
extra_query: dict | None
```

input

Python

```
input: str | list[str] | list[int] | list[list[int]] | None
```

timeout

Python

```
timeout: float | None
```

user

Python

```
user: str | None
```

OpenAIPromptExecutionSettings Class

Reference

Common request settings for (Azure) OpenAI services.

Inheritance [PromptExecutionSettings](#) → OpenAIPromptExecutionSettings

Constructor

Python

```
OpenAIPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
extension_data	
Required*	
ai_model_id	
Required*	
frequency_penalty	
Required*	
logit_bias	

Name	Description
Required*	
max_tokens	
Required*	
number_of_responses	
Required*	
presence_penalty	
Required*	
seed	
Required*	
stop	
Required*	
stream	
Required*	
temperature	
Required*	
top_p	
Required*	
user	
Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [`ConfigDict`] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='model'), 'extension_data':
FieldInfo(annotation=dict[str, Any], required=False,
default_factory=dict), 'frequency_penalty':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]), 'logit_bias':
FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],
required=False, default=None), 'max_tokens':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
metadata=[Gt(gt=0)]), 'number_of_responses':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),
Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,
NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),
Le(le=2.0)]), 'seed': FieldInfo(annotation=Union[int, NoneType],
required=False, default=None), 'service_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
metadata=[MinLen(min_length=1)]), 'stop': FieldInfo(annotation=Union[str,
list[str], NoneType], required=False, default=None), 'stream':
FieldInfo(annotation=bool, required=False, default=False), 'temperature':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=0.0), Le(le=2.0)]), 'top_p':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

extension_data

Python

```
extension_data: dict[str, Any]
```

frequency_penalty

Python

```
frequency_penalty: float | None
```

logit_bias

Python

```
logit_bias: dict[str | int, float] | None
```

max_tokens

Python

```
max_tokens: int | None
```

number_of_responses

Python

```
number_of_responses: int | None
```

presence_penalty

Python

```
presence_penalty: float | None
```

seed

Python

```
seed: int | None
```

service_id

Python

```
service_id: str | None
```

stop

Python

```
stop: str | list[str] | None
```

stream

Python

```
stream: bool
```

temperature

Python

```
temperature: float | None
```

top_p

Python

```
top_p: float | None
```

user

Python

```
user: str | None
```

OpenAITextPromptExecutionSettings Class

Reference

Specific settings for the completions endpoint.

Inheritance [OpenAIPromptExecutionSettings](#) → OpenAITextPromptExecutionSettings

Constructor

Python

```
OpenAITextPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
prompt: str | None = None, best_of: int | None = None, echo: bool = False,  
logprobs: int | None = None, suffix: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
service_id	default value: None

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
extension_data Required*	
ai_model_id Required*	

Name	Description
frequency_penalty Required*	
logit_bias Required*	
max_tokens Required*	
number_of_responses Required*	
presence_penalty Required*	
seed Required*	
stop Required*	
stream Required*	
temperature Required*	
top_p Required*	
user Required*	
prompt Required*	
best_of Required*	
echo Required*	
logprobs Required*	
suffix Required*	

Methods

[Expand table](#)

<code>check_best_of_and_n</code>	Check that the <code>best_of</code> parameter is not greater than the <code>number_of_responses</code> parameter.
----------------------------------	---

check_best_of_and_n

Check that the `best_of` parameter is not greater than the `number_of_responses` parameter.

Python	<code>check_best_of_and_n() -> OpenAITextPromptExecutionSettings</code>
--------	--

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python	<code>model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}</code>
--------	---

model_config

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][`pydantic.config.ConfigDict`].

Python	<code>model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}</code>
--------	--

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'ai_model_id':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        alias_priority=2, serialization_alias='model'),  
    'best_of':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Ge(ge=1)]),  
    'echo': FieldInfo(annotation=bool, required=False, default=False,  
        default=False),  
    'extension_data': FieldInfo(annotation=dict[str, Any],  
        required=False, default_factory=dict),  
    'frequency_penalty':  
        FieldInfo(annotation=Union[float, NoneType], required=False,  
        default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]),  
    'logit_bias':  
        FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],  
        required=False, default=None),  
    'logprobs':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Ge(ge=0), Le(le=5)]),  
    'max_tokens':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Gt(gt=0)]),  
    'number_of_responses':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),  
        Le(le=128)]),  
    'presence_penalty': FieldInfo(annotation=Union[float,  
        NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),  
        Le(le=2.0)]),  
    'prompt': FieldInfo(annotation=Union[str, NoneType],  
        required=False, default=None),  
    'seed': FieldInfo(annotation=Union[int,  
        NoneType], required=False, default=None),  
    'service_id':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        metadata=[MinLen(min_length=1)]),  
    'stop': FieldInfo(annotation=Union[str,  
        list[str], NoneType], required=False, default=None),  
    'stream':  
        FieldInfo(annotation=bool, required=False, default=False),  
    'suffix':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'temperature': FieldInfo(annotation=Union[float, NoneType],  
        required=False, default=None, metadata=[Ge(ge=0.0), Le(le=2.0)]),  
    'top_p': FieldInfo(annotation=Union[float, NoneType], required=False,  
        default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]),  
    'user':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}  
}
```

ai_model_id

Python

```
ai_model_id: str | None
```

best_of

Python

```
best_of: int | None
```

echo

Python

```
echo: bool
```

extension_data

Python

```
extension_data: dict[str, Any]
```

frequency_penalty

Python

```
frequency_penalty: float | None
```

logit_bias

Python

```
logit_bias: dict[str | int, float] | None
```

logprobs

Python

```
logprobs: int | None
```

max_tokens

Python

```
max_tokens: int | None
```

number_of_responses

Python

```
number_of_responses: int | None
```

presence_penalty

Python

```
presence_penalty: float | None
```

prompt

Python

```
prompt: str | None
```

seed

Python

```
seed: int | None
```

service_id

Python

```
service_id: str | None
```

stop

Python

```
stop: str | list[str] | None
```

stream

Python

```
stream: bool
```

suffix

Python

```
suffix: str | None
```

temperature

Python

```
temperature: float | None
```

top_p

Python

```
top_p: float | None
```

user

Python

```
user: str | None
```

azure_chat_completion Module

Reference

Classes

Expand table

[AzureChatCompletion](#)

Azure Chat completion class.

Initialize an AzureChatCompletion service.

AzureChatCompletion Class

Reference

Azure Chat completion class.

Initialize an AzureChatCompletion service.

Inheritance [AzureOpenAIConfigBase](#) → [AzureChatCompletion](#)

[OpenAIChatCompletionBase](#) → [AzureChatCompletion](#)

[OpenAITextCompletionBase](#) → [AzureChatCompletion](#)

Constructor

Python

```
AzureChatCompletion(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
None} Required*	<xref:<xref:ad_token {str} > The service ID for the Azure deployment. (Optional)
None} Required*	The optional api key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional deployment. If provided, will override the value (chat_deployment_name) in the env vars or .env file.
None} Required*	The optional deployment endpoint. If provided will override the value in the env vars or .env file.
None} Required*	The optional deployment base_url. If provided will override the value in the env vars or .env file.
None}	

Name	Description
Required*	The optional deployment api version. If provided will override the value in the env vars or .env file.
None} Required*	The Azure Active Directory token. (Optional)
{AsyncAzureADTokenProvider} Required*	<xref:ad_token_provider> The Azure Active Directory token provider. (Optional)
{Mapping[str] Required*	str (<xref:default_headers> The default headers mapping of string keys to string values for HTTP requests. (Optional)
str]} Required*	The default headers mapping of string keys to string values for HTTP requests. (Optional)
use. Required*	<xref:<xref:async_client {AsyncAzureOpenAI}> <xref:None} -- An existing client to>>
vars. Required*	<xref:<xref:env_file_path {str}> <xref:None} -- Use the environment settings file as a fallback to using env>>
service_id	default value: None
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
ad_token	default value: None
ad_token_provider	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

<code>from_dict</code>	Initialize an Azure OpenAI service from a dictionary of settings.
<code>get_prompt_execution_settings_class</code>	Create a request settings object.
<code>split_message</code>	<p>Split a Azure On Your Data response into separate ChatMessageContents.</p> <p>If the message does not have three contents, and those three are one each of: FunctionCallContent, FunctionResultContent, and TextContent, it will not return three messages, potentially only one or two.</p> <p>The order of the returned messages is as expected by OpenAI.</p>

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureChatCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: service_id, and optionally: ad_auth, ad_token_provider, default_headers

get_prompt_execution_settings_class

Create a request settings object.

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

split_message

Split a Azure On Your Data response into separate ChatMessageContents.

If the message does not have three contents, and those three are one each of: FunctionCallContent, FunctionResultContent, and TextContent, it will not return three messages, potentially only one or two.

The order of the returned messages is as expected by OpenAI.

Python

```
static split_message(message: ChatMessageContent) ->
list[semantic_kernel.contents.chat_message_content.ChatMessageContent]
```

Parameters

[\[\] Expand table](#)

Name	Description
message Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
            FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
                FieldInfo(annotation=int, required=False, default=0), 'service_id':  
                    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
                        FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

Python

```
service_id: str
```

total_tokens

Python

```
total_tokens: int
```

azure_config_base Module

Reference

Classes

[] [Expand table](#)

[AzureOpenAIConfigBase](#) Internal class for configuring a connection to an Azure OpenAI service.

Internal class for configuring a connection to an Azure OpenAI service.

The *validate_call* decorator is used with a configuration that allows arbitrary types. This is necessary for types like *HttpsUrl* and *OpenAIModelTypes*.

AzureOpenAIConfigBase Class

Reference

Internal class for configuring a connection to an Azure OpenAI service.

Internal class for configuring a connection to an Azure OpenAI service.

The *validate_call* decorator is used with a configuration that allows arbitrary types. This is necessary for types like *HttpsUrl* and *OpenAIModelTypes*.

Inheritance [OpenAIHandler](#) → AzureOpenAIConfigBase

Constructor

Python

```
AzureOpenAIConfigBase(deployment_name: str, ai_model_type: OpenAIModelTypes,  
endpoint: Url | None = None, base_url: Url | None = None, api_version: str =  
'2024-02-01', service_id: str | None = None, api_key: str | None = None,  
ad_token: str | None = None, ad_token_provider: Callable[[], str |  
Awaitable[str]] | None = None, default_headers: Mapping[str, str] | None =  
None, async_client: AsyncAzureOpenAI | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
deployment. Required*	<xref:endpoint {Optional}>[<xref:HttpsUrl>]<xref:> -- The specific endpoint URL for the>
deploy. Required*	<xref:><xref:ai_model_type {OpenAIModelTypes}> -- The type> of <xref:OpenAI model to>>
deployment. Required*	
services. Required*	<xref:api_key {Optional}>[str]<xref:> -- API key for Azure>
DEFAULT_AZURE_API_VERSION. Required*	<xref:><xref:api_version {str}> -- Azure API version. Defaults to the defined>>
services.	

Name	Description
Required*	
authentication. Required*	<xref:ad_token {Optional}>[str]<xref:} -- Azure AD token for>
{Optional[Callable[]] Required*	Callable[] (<xref:ad_token_provider> or coroutine function providing Azure AD tokens. (Optional)
Union[str Required*	or coroutine function providing Azure AD tokens. (Optional)
callable Required*	Awaitable[str]]]]<xref:} -- A> or coroutine function providing Azure AD tokens. (Optional)
{Union[Mapping[str Required*	Mapping[str (<xref:default_headers>
str] Required*	(Optional
requests. Required*	None]<xref:} -- Default headers for HTTP>
use. Required*	<xref:async_client {Optional}>[<xref:AsyncAzureOpenAI>]<xref:} -- An existing client to>
deployment_name Required*	
ai_model_type Required*	
endpoint	default value: None
base_url	default value: None
api_version	default value: 2024-02-01
service_id	default value: None
api_key	default value: None
ad_token	default value: None
ad_token_provider	default value: None
default_headers	default value: None
async_client	default value: None

Methods

 Expand table

to_dict

Python

```
to_dict() -> dict[str, str]
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_type':  
    FieldInfo(annotation=OpenAISchemaTypes, required=False, default=  
        <OpenAISchemaTypes.CHAT: 'chat'>), 'client':  
    FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'total_tokens':  
    FieldInfo(annotation=int, required=False, default=0)}
```

azure_text_completion Module

Reference

Classes

[+] Expand table

[AzureTextCompletion](#)

Azure Text Completion class.

Initialize an AzureTextCompletion service.

AzureTextCompletion Class

Reference

Azure Text Completion class.

Initialize an AzureTextCompletion service.

Inheritance [AzureOpenAIConfigBase](#) → [AzureTextCompletion](#)
[OpenAITextCompletionBase](#) → [AzureTextCompletion](#)

Constructor

Python

```
AzureTextCompletion(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
service_id	The service ID for the Azure deployment. (Optional) default value: None
None} Required*	<xref:<xref:api_version {str}> > The optional api key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file.
None} Required*	The optional deployment endpoint. If provided will override the value in the env vars or .env file.
None} Required*	The optional deployment base_url. If provided will override the value in the env vars or .env file.
None}	

Name	Description
Required*	The optional deployment api version. If provided will override the value in the env vars or .env file.
ad_token	The Azure Active Directory token. (Optional) default value: None
ad_token_provider	The Azure Active Directory token provider. (Optional) default value: None
default_headers	The default headers mapping of string keys to string values for HTTP requests. (Optional) default value: None
use. Required*	<xref:async_client {Optional}> [<xref:AsyncAzureOpenAI>] <xref:> -- An existing client to>
to Required*	<xref:><xref:env_file_path {str} <xref:None}> -- Use the environment settings file as a fallback>> environment variables. (Optional)
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Azure OpenAI service from a dictionary of settings.
------------------	---

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureTextCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: deployment_name, endpoint, api_key and optionally: api_version, ad_auth

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
        FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
        FieldInfo(annotation=int, required=False, default=0), 'service_id':  
        FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
        FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

```
Python
```

```
service_id: str
```

total_tokens

```
Python
```

```
total_tokens: int
```

azure_text_embedding Module

Reference

Classes

 Expand table

AzureTextEmbedding	Azure Text Embedding class. Note: This class is experimental and may change in the future. Initialize an AzureTextEmbedding service. service_id: The service ID. (Optional) api_key {str None}: The optional api key. If provided, will override the value in the env vars or .env file. deployment_name {str None}: The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file. endpoint {str None}: The optional deployment endpoint. If provided will override the value in the env vars or .env file. base_url {str None}: The optional deployment base_url. If provided will override the value in the env vars or .env file. api_version {str None}: The optional deployment api version. If provided will override the value in the env vars or .env file. ad_token {str None}: The Azure AD token for authentication. (Optional) ad_auth {AsyncAzureADTokenProvider None}: Whether to use Azure Active Directory authentication. (Optional) The default value is False. default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional) async_client {Optional[AsyncAzureOpenAI]} – An existing client to use. (Optional) env_file_path {str None} – Use the environment settings file as a fallback to environment variables. (Optional)
------------------------------------	---

AzureTextEmbedding Class

Reference

Azure Text Embedding class.

Note: This class is experimental and may change in the future.

Initialize an AzureTextEmbedding service.

service_id: The service ID. (Optional) api_key {str | None}: The optional api key. If provided, will override the value in the

env vars or .env file.

deployment_name {str | None}: The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file.

endpoint {str | None}: The optional deployment endpoint. If provided will override the value in the env vars or .env file.

base_url {str | None}: The optional deployment base_url. If provided will override the value in the env vars or .env file.

api_version {str | None}: The optional deployment api version. If provided will override the value in the env vars or .env file.

ad_token {str | None}: The Azure AD token for authentication. (Optional) ad_auth {AsyncAzureADTokenProvider | None}: Whether to use Azure Active Directory authentication.

(Optional) The default value is False.

default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

async_client {Optional[AsyncAzureOpenAI]} – An existing client to use. (Optional)
env_file_path {str | None} – Use the environment settings file as a fallback to

environment variables. (Optional)

Inheritance [AzureOpenAIConfigBase](#) → AzureTextEmbedding

[OpenAITextEmbeddingBase](#) → AzureTextEmbedding

Constructor

Python

```
AzureTextEmbedding(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
ad_token	default value: None
ad_token_provider	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

from_dict	Initialize an Azure OpenAI service from a dictionary of settings.
-----------	---

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureTextEmbedding
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: deployment_name, endpoint, api_key and optionally: api_version, ad_auth

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
    FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'service_id':  
    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
    FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
    min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

is_experimental

Python

```
is_experimental = True
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

Python

```
service_id: str
```

total_tokens

Python

```
total_tokens: int
```

open_ai_chat_completion Module

Reference

Classes

[] [Expand table](#)

[OpenAIChatCompletion](#) OpenAI Chat completion class.

Initialize an OpenAIChatCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to

string values for HTTP requests. (Optional)

OpenAIChatCompletion Class

Reference

OpenAI Chat completion class.

Initialize an OpenAIChatCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

Inheritance [OpenAIConfigBase](#) → OpenAIChatCompletion

[OpenAIChatCompletionBase](#) → OpenAIChatCompletion

[OpenAITextCompletionBase](#) → OpenAIChatCompletion

Constructor

Python

```
OpenAIChatCompletion(ai_model_id: str | None = None, service_id: str | None = None, api_key: str | None = None, org_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] Expand table

Name	Description
name Required*	<xref:<xref:ai_model_id {str} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>
override Required*	<xref:<xref:api_key {str} <xref:None} -- The optional API key to use. If provided will>>

Name	Description
	the env vars or .env file value.
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
fallback Required*	<xref:><xref:env_file_path {str}> <xref:None> -- Use the environment settings file as a>> to environment variables. (Optional)
ai_model_id	default value: None
service_id	default value: None
api_key	default value: None
org_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Open AI service from a dictionary of settings.
---------------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> OpenAIChatCompletion
```

Parameters

[\[\] Expand table](#)

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':
```

```
    FieldInfo(annotation=int, required=False, default=0), 'service_id':  
    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
    FieldInfo(annotation=int, required=False, default=0)})
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

Python

```
service_id: str
```

total_tokens

Python

```
total_tokens: int
```

open_ai_chat_completion_base Module

Reference

Classes

[+] Expand table

InvokeTermination	Exception for termination of function invocation.
OpenAIChatCompletionBase	<p>OpenAI Chat completion class.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [<i>ValidationError</i>][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>

InvokeTermination Class

Reference

Exception for termination of function invocation.

Inheritance builtins.Exception → InvokeTermination

Constructor

Python

```
InvokeTermination()
```

OpenAIChatCompletionBase Class

Reference

OpenAI Chat completion class.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [OpenAIHandler](#) → OpenAIChatCompletionBase
[ChatCompletionClientBase](#) → OpenAIChatCompletionBase

Constructor

Python

```
OpenAIChatCompletionBase(*, ai_model_id: str, service_id: str = '', client:  
    AsyncOpenAI, ai_model_type: OpenAIModelTypes = OpenAIModelTypes.CHAT,  
    prompt_tokens: int = 0, completion_tokens: int = 0, total_tokens: int = 0)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
ai_model_id Required*	
service_id Required*	
client Required*	
ai_model_type	default value: OpenAIModelTypes.CHAT
prompt_tokens Required*	
completion_tokens	

Name	Description
Required*	
total_tokens	
Required*	

Methods

[+] Expand table

get_chat_message_contents	Executes a chat completion request and returns the result.
get_prompt_execution_settings_class	Create a request settings object.
get_streaming_chat_message_contents	Executes a streaming chat completion request and returns the result.

get_chat_message_contents

Executes a chat completion request and returns the result.

Python

```
async get_chat_message_contents(chat_history: ChatHistory, settings: OpenAIChatPromptExecutionSettings, **kwargs: Any) -> list[semantic_kernel.contents.chat_message_content.ChatMessageContent]
```

Parameters

[+] Expand table

Name	Description
completion. Required*	<xref:<xref:chat_history {ChatHistory} -- The chat history to use for the chat>>
use Required*	<xref:<xref:settings {OpenAIChatPromptExecutionSettings} <xref:AzureChatPromptExecutionSettings} -- The settings to>> for the chat completion request.
{Dict[str Required*	str (<xref:kwargs>

Name	Description
arguments. Required*	<code>Any]</code> <xref:-- The optional>
chat_history Required*	
settings Required*	

Returns

[] Expand table

Type	Description
	List[ChatMessageContent] – The completion result(s).

get_prompt_execution_settings_class

Create a request settings object.

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

get_streaming_chat_message_contents

Executes a streaming chat completion request and returns the result.

Python

```
async get_streaming_chat_message_contents(chat_history: ChatHistory,
settings: OpenAIChatPromptExecutionSettings, **kwargs: Any) ->
AsyncGenerator[list[semantic_kernel.contents.streaming_chat_message_content.StreamingChatMessageContent | None], Any]
```

Parameters

[] Expand table

Name	Description
completion. Required*	<xref:<xref:chat_history {ChatHistory} -- The chat history to use for the chat>>
use Required*	<xref:<xref:settings {OpenAIChatPromptExecutionSettings} <xref:AzureChatPromptExecutionSettings} -- The settings to>> for the chat completion request.
{Dict[str Required*	str (<xref:kwargs>
arguments. Required*	Any]<xref:} -- The optional>
chat_history Required*	
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
            FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
                FieldInfo(annotation=int, required=False, default=0), 'service_id':  
                    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
                        FieldInfo(annotation=int, required=False, default=0)}
```

open_ai_config_base Module

Reference

Classes

[+] Expand table

OpenAIConfigBase	Initialize a client for OpenAI services.
----------------------------------	--

This constructor sets up a client to interact with OpenAI's API, allowing for different types of AI model interactions, like chat or text completion.

OpenAIConfigBase Class

Reference

Initialize a client for OpenAI services.

This constructor sets up a client to interact with OpenAI's API, allowing for different types of AI model interactions, like chat or text completion.

Inheritance [OpenAIHandler](#) → OpenAIConfigBase

Constructor

Python

```
OpenAIConfigBase(ai_model_id: str = FieldInfo(annotation=str, required=True,
metadata=[MinLen(min_length=1)]), api_key: str | None =
FieldInfo(annotation=Union[str, NoneType], required=True, metadata=
[MinLen(min_length=1)]), ai_model_type: OpenAIModelTypes | None =
OpenAIModelTypes.CHAT, org_id: str | None = None, service_id: str | None =
None, default_headers: Mapping[str, str] | None = None, async_client:
AsyncOpenAI | None = None)
```

Parameters

[Expand table](#)

Name	Description
non-empty. Required*	<xref:<xref:ai_model_id {str} -- OpenAI model identifier. Must be>> Default to a preset value.
authentication. Required*	<xref:api_key {Optional}>[str]<xref:> -- OpenAI API key for> Must be non-empty. (Optional)
OpenAI Required*	<xref:ai_model_type {Optional}>[OpenAIModelTypes]<xref:> -- The type of> model to interact with. Defaults to CHAT.
optional Required*	<xref:org_id {Optional}>[str]<xref:> -- OpenAI organization ID. This is> unless the account belongs to multiple organizations.
{Optional[Mapping[str Required*	Mapping[str (<xref:default_headers> for HTTP requests. (Optional)

Name	Description
headers Required*	<code>str]]<xref:} -- Default></code> for HTTP requests. (Optional)
ai_model_id	default value: annotation=str required=True metadata=[MinLen(min_length=1)]
api_key	default value: annotation=Union[str, NoneType] required=True metadata=[MinLen(min_length=1)]
ai_model_type	default value: OpenAIModelTypes.CHAT
org_id	default value: None
service_id	default value: None
default_headers	default value: None
async_client	default value: None

Methods

[\[\] Expand table](#)

to_dict	Create a dict of the service settings.
-------------------------	--

to_dict

Create a dict of the service settings.

Python

```
to_dict() -> dict[str, str]
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_type':  
FieldInfo(annotation=OpenAIModelTypes, required=False, default=  
<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'total_tokens':  
FieldInfo(annotation=int, required=False, default=0)}
```

open_ai_handler Module

Reference

Classes

 [Expand table](#)

[OpenAIHandler](#) Internal class for calls to OpenAI API's.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

OpenAIHandler Class

Reference

Internal class for calls to OpenAI API's.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → OpenAIHandler

[ABC](#) → OpenAIHandler

Constructor

Python

```
OpenAIHandler(*, client: AsyncOpenAI, ai_model_type: OpenAIModelTypes =  
    OpenAIModelTypes.CHAT, prompt_tokens: int = 0, completion_tokens: int = 0,  
    total_tokens: int = 0)
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
client Required*	
ai_model_type	default value: OpenAIModelTypes.CHAT
prompt_tokens Required*	
completion_tokens Required*	
total_tokens Required*	

Methods

[Expand table](#)

<code>store_usage</code>

store_usage

Python

<code>store_usage(response)</code>

Parameters

[Expand table](#)

Name	Description
<code>response</code> Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

<code>model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}</code>

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_type':  
FieldInfo(annotation=OpenAIModelTypes, required=False, default=  
<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'total_tokens':  
FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

prompt_tokens

Python

```
prompt_tokens: int
```

total_tokens

Python

```
total_tokens: int
```

open_ai_model_types Module

Reference

Enums

[+] Expand table

OpenAIModelTypes	OpenAI model types, can be text, chat or embedding.
----------------------------------	---

OpenAIModelTypes Enum

Reference

OpenAI model types, can be text, chat or embedding.

Inheritance [Enum](#) → OpenAIModelTypes

Constructor

Python

```
OpenAIModelTypes(value, names=None, *, module=None, qualname=None,  
type=None, start=1, boundary=None)
```

Fields

[] [Expand table](#)

CHAT
EMBEDDING
TEXT

open_ai_text_completion Module

Reference

Classes

[] [Expand table](#)

[OpenAITextCompletion](#) OpenAI Text Completion class.

Initialize an OpenAITextCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to

string values for HTTP requests. (Optional)

OpenAITextCompletion Class

Reference

OpenAI Text Completion class.

Initialize an OpenAITextCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

Inheritance [OpenAITextCompletionBase](#) → OpenAITextCompletion
[OpenAIConfigBase](#) → OpenAITextCompletion

Constructor

Python

```
OpenAITextCompletion(ai_model_id: str | None = None, api_key: str | None = None, org_id: str | None = None, service_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] [Expand table](#)

Name	Description
name Required*	<xref:<xref:ai_model_id {str} <xref:None} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>
override Required*	<xref:<xref:api_key {str} <xref:None} -- The optional API key to use. If provided will>> the env vars or .env file value.

Name	Description
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
to Required*	<xref:><xref:env_file_path {str}> <xref:None> -- Use the environment settings file as a fallback>> environment variables. (Optional)
ai_model_id	default value: None
api_key	default value: None
org_id	default value: None
service_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

from_dict	Initialize an Open AI service from a dictionary of settings.
------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> OpenAITextCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':
FieldInfo(annotation=int, required=False, default=0), 'service_id':
FieldInfo(annotation=str, required=False, default=''), 'total_tokens':
FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

Python

```
service_id: str
```

total_tokens

Python

```
total_tokens: int
```

open_ai_text_completion_base Module

Reference

Classes

 Expand table

<p>OpenAITextCompletionBase</p>	Create a new model by parsing and validating input data from keyword arguments.
---	---

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

OpenAITextCompletionBase Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [OpenAIHandler](#) → OpenAITextCompletionBase
[TextCompletionClientBase](#) → OpenAITextCompletionBase

Constructor

Python

```
OpenAITextCompletionBase(*, ai_model_id: str, service_id: str = '', client:  
    AsyncOpenAI, ai_model_type: OpenAIModelTypes = OpenAIModelTypes.CHAT,  
    prompt_tokens: int = 0, completion_tokens: int = 0, total_tokens: int = 0)
```

Keyword-Only Parameters

[] [Expand table](#)

Name	Description
ai_model_id Required*	
service_id Required*	
client Required*	
ai_model_type	default value: OpenAIModelTypes.CHAT
prompt_tokens Required*	
completion_tokens Required*	

Name	Description
total_tokens Required*	

Methods

[+] Expand table

get_prompt_execution_settings_class	Create a request settings object.
get_streaming_text_contents	Executes a completion request and streams the result. Supports both chat completion and text completion.
get_text_contents	Executes a completion request and returns the result.

get_prompt_execution_settings_class

Create a request settings object.

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

get_streaming_text_contents

Executes a completion request and streams the result. Supports both chat completion and text completion.

Python

```
async get_streaming_text_contents(prompt: str, settings: OpenAIPromptExecutionSettings) -> AsyncGenerator[list['StreamingTextContent'], Any]
```

Parameters

[+] Expand table

Name	Description
request. Required*	<xref:<xref:settings {OpenAITextPromptExecutionSettings} -- The settings to use for the completion>>
request. Required*	
prompt Required*	
settings Required*	

get_text_contents

Executes a completion request and returns the result.

Python

```
async get_text_contents(prompt: str, settings:  
OpenAIPromptExecutionSettings) -> list[ 'TextContent' ]
```

Parameters

[] [Expand table](#)

Name	Description
request. Required*	<xref:<xref:settings {OpenAITextPromptExecutionSettings} -- The settings to use for the completion>>
request. Required*	
prompt Required*	
settings Required*	

Returns

[] [Expand table](#)

Type	Description
	List["TextContent"] – The completion result(s).

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':
FieldInfo(annotation=int, required=False, default=0), 'service_id':
```

```
FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
FieldInfo(annotation=int, required=False, default=0)}
```

open_ai_text_embedding Module

Reference

Classes

 Expand table

[OpenAITextEmbedding](#) OpenAI Text Embedding class.

Note: This class is experimental and may change in the future.

Initializes a new instance of the OpenAITextCompletion class.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers {Optional[Mapping[str: The default headers mapping of string keys to string values for HTTP requests. (Optional)

OpenAITextEmbedding Class

Reference

OpenAI Text Embedding class.

Note: This class is experimental and may change in the future.

Initializes a new instance of the OpenAITextCompletion class.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers {Optional[Mapping[str: The default headers mapping of string keys to

string values for HTTP requests. (Optional)

Inheritance [OpenAIConfigBase](#) → OpenAITextEmbedding

[OpenAITextEmbeddingBase](#) → OpenAITextEmbedding

Constructor

Python

```
OpenAITextEmbedding(ai_model_id: str, api_key: str | None = None, org_id: str | None = None, service_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] [Expand table](#)

Name	Description
name Required*	<xref:<xref:ai_model_id {str} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>

Name	Description
override Required*	<xref:><xref:api_key {str} <xref:None> -- The optional API key to use. If provided will>> the env vars or .env file value.
str]]} Required*	The default headers mapping of string keys to string values for HTTP requests. (Optional)
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
as Required*	<xref:><xref:env_file_path {str} <xref:None> -- Use the environment settings file>> a fallback to environment variables. (Optional)
ai_model_id Required*	
api_key	default value: None
org_id	default value: None
service_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Open AI service from a dictionary of settings.
---------------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> OpenAITextEmbedding
```

Parameters

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
```

```
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'service_id':  
FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
FieldInfo(annotation=int, required=False, default=0)}
```

ai_model_id

Python

```
ai_model_id: Annotated[str, StringConstraints(strip_whitespace=True,  
min_length=1)]
```

ai_model_type

Python

```
ai_model_type: OpenAIModelTypes
```

client

Python

```
client: AsyncOpenAI
```

completion_tokens

Python

```
completion_tokens: int
```

is_experimental

Python

```
is_experimental = True
```

prompt_tokens

Python

```
prompt_tokens: int
```

service_id

Python

```
service_id: str
```

total_tokens

Python

```
total_tokens: int
```

open_ai_text_embedding_base Module

Reference

Classes

 Expand table

[OpenAITextEmbeddingBase](#) Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

OpenAITextEmbeddingBase Class

Reference

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [OpenAIHandler](#) → OpenAITextEmbeddingBase

[EmbeddingGeneratorBase](#) → OpenAITextEmbeddingBase

Constructor

Python

```
OpenAITextEmbeddingBase(*, ai_model_id: str, service_id: str = '', client:  
    AsyncOpenAI, ai_model_type: OpenAIModelTypes = OpenAIModelTypes.CHAT,  
    prompt_tokens: int = 0, completion_tokens: int = 0, total_tokens: int = 0)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
ai_model_id Required*	
service_id Required*	
client Required*	
ai_model_type	default value: <code>OpenAIModelTypes.CHAT</code>
prompt_tokens Required*	
completion_tokens	

Name	Description
Required*	
total_tokens	
Required*	

Methods

[\[\] Expand table](#)

generate_embeddings	Generates embeddings for the given texts. :param : see OpenAIEmbeddingPromptExecutionSettings for the details.
get_prompt_execution_settings_class	

generate_embeddings

Generates embeddings for the given texts.

:param : see OpenAIEmbeddingPromptExecutionSettings for the details.

Python

```
async generate_embeddings(texts: list[str], batch_size: int | None = None, **kwargs: Any) -> ndarray
```

Parameters

[\[\] Expand table](#)

Name	Description
texts	
Required*	
batch_size	default value: None

Returns

Type	Description
	ndarray – The embeddings for the text.

get_prompt_execution_settings_class

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
            FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
                FieldInfo(annotation=int, required=False, default=0), 'service_id':  
                    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
                        FieldInfo(annotation=int, required=False, default=0)}
```

is_experimental

Python

```
is_experimental = True
```

utils Module

Reference

Functions

kernel_function_metadata_to_openai_tool_format

Convert the kernel function metadata to OpenAI format.

Python

```
kernel_function_metadata_to_openai_tool_format(metadata:  
KernelFunctionMetadata) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
metadata Required*	

update_settings_from_function_call_configuration

Update the settings from a FunctionCallConfiguration.

Python

```
update_settings_from_function_call_configuration(function_call_configurat  
ion: FunctionCallConfiguration, settings:  
OpenAIChatPromptExecutionSettings) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
function_call_configuration	

Name	Description
Required*	
settings Required*	

azure_open_ai_settings Module

Reference

Classes

 Expand table

[AzureOpenAISettings](#) AzureOpenAI model settings

The settings are first loaded from environment variables with the prefix '`'>>AZURE_OPENAI_<<'`'. If the environment variables are not found, the settings can be loaded from a .env file with the encoding 'utf-8'. If the settings are not found in the .env file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '`'>>AZURE_OPENAI_<<'`' are:

- `chat_deployment_name`: str - The name of the Azure Chat deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var AZURE_OPENAI_CHAT_DEPLOYMENT_NAME)
- `text_deployment_name`: str - The name of the Azure Text deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var AZURE_OPENAI_TEXT_DEPLOYMENT_NAME)
- `embedding_deployment_name`: str - The name of the Azure Embedding deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var AZURE_OPENAI_EMBEDDING_DEPLOYMENT_NAME)
- `api_key`: SecretStr - The API key for the Azure deployment. This value can be found in the Keys & Endpoint section when examining your resource in the Azure portal. You can use either KEY1 or KEY2. (Env

```
var AZURE_OPENAI_API_KEY)
```

- base_url: HttpsUrl | None - base_url: The url of the Azure deployment. This value

can be found in the Keys & Endpoint section when examining your resource from the Azure portal, the base_url consists of the endpoint, followed by /openai/deployments/{deployment_name}/, use endpoint if you only want to supply the endpoint. (Env var AZURE_OPENAI_BASE_URL)

- endpoint: HttpsUrl - The endpoint of the Azure deployment. This value

can be found in the Keys & Endpoint section when examining your resource from the Azure portal, the endpoint should end in openai.azure.com. If both base_url and endpoint are supplied, base_url will be used. (Env var AZURE_OPENAI_ENDPOINT)

- api_version: str | None - The API version to use. The default value is "2024-02-01".

(Env var AZURE_OPENAI_API_VERSION)

- env_file_path: str | None - if provided, the .env settings are read from this file path location

AzureOpenAISettings Class

Reference

AzureOpenAI model settings

The settings are first loaded from environment variables with the prefix '`'>>AZURE_OPENAI_<<`'. If the environment variables are not found, the settings can be loaded from a .env file with the encoding 'utf-8'. If the settings are not found in the .env file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '`'>>AZURE_OPENAI_<<`' are:

- `chat_deployment_name`: str - The name of the Azure Chat deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var `AZURE_OPENAI_CHAT_DEPLOYMENT_NAME`)
- `text_deployment_name`: str - The name of the Azure Text deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var `AZURE_OPENAI_TEXT_DEPLOYMENT_NAME`)
- `embedding_deployment_name`: str - The name of the Azure Embedding deployment. This value will correspond to the custom name you chose for your deployment when you deployed a model. This value can be found under Resource Management > Deployments in the Azure portal or, alternatively, under Management > Deployments in Azure OpenAI Studio. (Env var `AZURE_OPENAI_EMBEDDING_DEPLOYMENT_NAME`)
- `api_key`: SecretStr - The API key for the Azure deployment. This value can be found in the Keys & Endpoint section when examining your resource in the Azure portal. You can use either KEY1 or KEY2. (Env var `AZURE_OPENAI_API_KEY`)
- `base_url`: HttpsUrl | None - `base_url`: The url of the Azure deployment. This value

can be found in the Keys & Endpoint section when examining your resource from the Azure portal, the base_url consists of the endpoint, followed by /openai/deployments/{deployment_name}/, use endpoint if you only want to supply the endpoint. (Env var AZURE_OPENAI_BASE_URL)

- endpoint: `HttpsUrl` - The endpoint of the Azure deployment. This value

can be found in the Keys & Endpoint section when examining your resource from the Azure portal, the endpoint should end in openai.azure.com. If both base_url and endpoint are supplied, base_url will be used. (Env var AZURE_OPENAI_ENDPOINT)

- api_version: `str | None` - The API version to use. The default value is "2024-02-01".
(Env var AZURE_OPENAI_API_VERSION)
- env_file_path: `str | None` - if provided, the .env settings are read from this file path location

Inheritance `pydantic_settings.main.BaseSettings` → `AzureOpenAISettings`

Constructor

Python

```
AzureOpenAISettings(_case_sensitive: bool | None = None, _env_prefix: str | None = None, _env_file: DotenvType | None = WindowsPath('.'), _env_file_encoding: str | None = None, _env_ignore_empty: bool | None = None, _env_nested_delimiter: str | None = None, _env_parse_none_str: str | None = None, _secrets_dir: str | Path | None = None, *, env_file_path: str | None = None, chat_deployment_name: str | None = None, text_deployment_name: str | None = None, embedding_deployment_name: str | None = None, endpoint: Url | None = None, base_url: Url | None = None, api_key: SecretStr | None = None, api_version: str | None = None)
```

Parameters

[] Expand table

Name	Description
<code>_case_sensitive</code>	default value: <code>None</code>
<code>_env_prefix</code>	default value: <code>None</code>
<code>_env_file</code>	default value: <code>.</code>

Name	Description
<code>_env_file_encoding</code>	default value: None
<code>_env_ignore_empty</code>	default value: None
<code>_env_nested_delimiter</code>	default value: None
<code>_env_parse_none_str</code>	default value: None
<code>_secrets_dir</code>	default value: None

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>env_file_path</code> Required*	
<code>chat_deployment_name</code> Required*	
<code>text_deployment_name</code> Required*	
<code>embedding_deployment_name</code> Required*	
<code>endpoint</code> Required*	
<code>base_url</code> Required*	
<code>api_key</code> Required*	
<code>api_version</code> Required*	

Methods

[\[\] Expand table](#)

<code>create</code>

create

Python

```
create(**kwargs)
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[SettingsConfigDict] = { 'arbitrary_types_allowed': True, 'case_sensitive': False, 'env_file': None, 'env_file_encoding': 'utf-8', 'env_ignore_empty': False, 'env_nested_delimiter': None, 'env_parse_none_str': None, 'env_prefix': 'AZURE_OPENAI_', 'extra': 'ignore', 'json_file': None, 'json_file_encoding': None, 'protected_namespaces': ('model_', 'settings_'), 'secrets_dir': None, 'toml_file': None, 'validate_default': True, 'yaml_file': None, 'yaml_file_encoding': None}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'api_key': FieldInfo(annotation=Union[SecretStr, NoneType], required=False,  
    default=None),  
    'api_version': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None),  
    'base_url': FieldInfo(annotation=Union[Annotated[Url, UrlConstraints(max_length=2083,  
    allowed_schemes=['https']), host_required=None, default_host=None,  
    default_port=None, default_path=None]], NoneType], required=False,  
    default=None),  
    'chat_deployment_name': FieldInfo(annotation=Union[str,  
    NoneType], required=False, default=None),  
    'embedding_deployment_name': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None),  
    'endpoint': FieldInfo(annotation=Union[Annotated[Url,  
    UrlConstraints(max_length=2083, allowed_schemes=['https']),  
    host_required=None, default_host=None, default_port=None,  
    default_path=None]], NoneType], required=False, default=None),  
    'env_file_path': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None),  
    'text_deployment_name': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None)}
```

api_key

Python

```
api_key: SecretStr | None
```

api_version

Python

```
api_version: str | None
```

base_url

Python

```
base_url: Url | None
```

chat_deployment_name

Python

```
chat_deployment_name: str | None
```

embedding_deployment_name

Python

```
embedding_deployment_name: str | None
```

endpoint

Python

```
endpoint: Url | None
```

env_file_path

Python

```
env_file_path: str | None
```

text_deployment_name

Python

```
text_deployment_name: str | None
```

Config Class

Reference

Inheritance builtins.object → Config

Constructor

Python

```
Config()
```

Attributes

case_sensitive

Python

```
case_sensitive = False
```

env_file

Python

```
env_file = None
```

env_file_encoding

Python

```
env_file_encoding = 'utf-8'
```

env_prefix

Python

```
env_prefix = 'AZURE_OPENAI_'
```

extra

Python

```
extra = 'ignore'
```

open_ai_settings Module

Reference

Classes

 Expand table

[OpenAISettings](#) OpenAI model settings

The settings are first loaded from environment variables with the prefix '`'>>OPENAI_<<`'. If the environment variables are not found, the settings can be loaded from a `.env` file with the encoding 'utf-8'. If the settings are not found in the `.env` file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '`'>>OPENAI_<<`' are:

- `api_key`: SecretStr - OpenAI API key, see
[https://platform.openai.com/account/api-keys ↗](https://platform.openai.com/account/api-keys)

(Env var OPENAI_API_KEY)
- `org_id`: str | None - This is usually optional unless your account belongs to multiple organizations.

(Env var OPENAI_ORG_ID)
- `chat_model_id`: str | None - The OpenAI chat model ID to use, for example, gpt-3.5-turbo or gpt-4.

(Env var OPENAI_CHAT_MODEL_ID)
- `text_model_id`: str | None - The OpenAI text model ID to use, for example, gpt-3.5-turbo-instruct.

(Env var OPENAI_TEXT_MODEL_ID)
- `embedding_model_id`: str | None - The OpenAI embedding model ID to use, for example, text-embedding-ada-002.

(Env var OPENAI_EMBEDDING_MODEL_ID)
- `env_file_path`: str | None - if provided, the `.env` settings are read from this file path location

OpenAISettings Class

Reference

OpenAI model settings

The settings are first loaded from environment variables with the prefix '>>OPENAI_<<'. If the environment variables are not found, the settings can be loaded from a .env file with the encoding 'utf-8'. If the settings are not found in the .env file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '>>OPENAI_<<' are:

- api_key: SecretStr - OpenAI API key, see <https://platform.openai.com/account/api-keys>
(Env var OPENAI_API_KEY)
- org_id: str | None - This is usually optional unless your account belongs to multiple organizations.
(Env var OPENAI_ORG_ID)
- chat_model_id: str | None - The OpenAI chat model ID to use, for example, gpt-3.5-turbo or gpt-4.
(Env var OPENAI_CHAT_MODEL_ID)
- text_model_id: str | None - The OpenAI text model ID to use, for example, gpt-3.5-turbo-instruct.
(Env var OPENAI_TEXT_MODEL_ID)
- embedding_model_id: str | None - The OpenAI embedding model ID to use, for example, text-embedding-ada-002.
(Env var OPENAI_EMBEDDING_MODEL_ID)
- env_file_path: str | None - if provided, the .env settings are read from this file path location

Inheritance `pydantic_settings.main.BaseSettings` → `OpenAISettings`

Constructor

Python

```
OpenAISettings(_case_sensitive: bool | None = None, _env_prefix: str | None = None, _env_file: DotenvType | None = WindowsPath('.'), _env_file_encoding: str | None = None, _env_ignore_empty: bool | None = None, _env_nested_delimiter: str | None = None, _env_parse_none_str: str | None = None, _secrets_dir: str | Path | None = None, *, env_file_path: str | None = None, org_id: str | None = None, api_key: SecretStr | None = None, chat_model_id: str | None = None, text_model_id: str | None = None, embedding_model_id: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
_case_sensitive	default value: None
_env_prefix	default value: None
_env_file	default value: .
_env_file_encoding	default value: None
_env_ignore_empty	default value: None
_env_nested_delimiter	default value: None
_env_parse_none_str	default value: None
_secrets_dir	default value: None

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
env_file_path	
Required*	
org_id	
Required*	
api_key	
Required*	
chat_model_id	

Name	Description
Required*	
text_model_id	
Required*	
embedding_model_id	
Required*	

Methods

 [Expand table](#)

[create](#)

Python

```
create(**kwargs)
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[SettingsConfigDict] = {'arbitrary_types_allowed': True, 'case_sensitive': False, 'env_file': None, 'env_file_encoding': 'utf-8', 'env_ignore_empty': False, 'env_nested_delimiter': None, 'env_parse_none_str': None, 'env_prefix': 'OPENAI_', 'extra': 'ignore', 'json_file': None, 'json_file_encoding': None, 'protected_namespaces': ('model_', 'settings_'), 'secrets_dir': None, 'toml_file': None, 'validate_default': True, 'yaml_file': None, 'yaml_file_encoding': None}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'api_key': FieldInfo(annotation=Union[SecretStr, NoneType], required=False, default=None), 'chat_model_id': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'embedding_model_id': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'env_file_path': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'org_id': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'text_model_id': FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

api_key

Python

```
api_key: SecretStr | None
```

chat_model_id

Python

```
chat_model_id: str | None
```

embedding_model_id

Python

```
embedding_model_id: str | None
```

env_file_path

Python

```
env_file_path: str | None
```

org_id

Python

```
org_id: str | None
```

text_model_id

Python

```
text_model_id: str | None
```

Config Class

Reference

Inheritance builtins.object → Config

Constructor

Python

```
Config()
```

Attributes

case_sensitive

Python

```
case_sensitive = False
```

env_file

Python

```
env_file = None
```

env_file_encoding

Python

```
env_file_encoding = 'utf-8'
```

env_prefix

Python

```
env_prefix = 'OPENAI_'
```

extra

Python

```
extra = 'ignore'
```

ApiKeyAuthentication Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [AzureChatRequestBase](#) → ApiKeyAuthentication

Constructor

Python

```
ApiKeyAuthentication(*, type: Literal['APIKey', 'api_key'] = 'api_key', key: str | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>type</code>	default value: <code>api_key</code>
<code>key</code> Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'key':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='key', serialization_alias='key'),  
    'type': FieldInfo(annotation=Literal['APIKey', 'api_key'],  
    required=False, default='api_key', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

key

Python

```
key: str | None
```

type

Python

```
type: Literal['APIKey', 'api_key']
```

AzureAIDataSource Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureAIDataSource

Constructor

Python

```
AzureAIDataSource(*, type: Literal['azure_search'] = 'azure_search',  
parameters: dict, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
type	default value: azure_search
parameters Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'parameters':  
    FieldInfo(annotation=dict, required=True, alias_priority=1,  
    validation_alias='parameters', serialization_alias='parameters',  
    metadata=[<class  
        'semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_ch  
        at_prompt_execution_settings.AzureAIDataSourceParameters'>]),  
    'type': FieldInfo(annotation=Literal['azure_search'], required=False,  
    default='azure_search', alias_priority=1, validation_alias='type',  
    serialization_alias='type')}
```

parameters

Python

```
parameters: dict
```

type

Python

```
type: Literal['azure_search']
```

AzureAIDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureDataSourceParameters](#) → AzureAIDataSourceParameters

Constructor

Python

```
AzureAIDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, endpoint: str | None = None, queryType: Literal['simple', 'semantic', 'vector', 'vectorSimpleHybrid', 'vectorSemanticHybrid'] = 'simple', authentication: ApiKeyAuthentication | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True

Name	Description
topNDocuments	default value: 5
semanticConfiguration Required*	
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	
endpoint Required*	
queryType	default value: simple
authentication Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,
```

```
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'authentication':  
        FieldInfo(annotation=Union[ApiKeyAuthentication, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='authentication', serialization_alias='authentication'),  
    'embedding_dependency':  
        FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='embeddingDependency',  
                  serialization_alias='embedding_dependency'),  
    'endpoint':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='endpoint',  
                  serialization_alias='endpoint'),  
    'fields_mapping':  
        FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
                  required=False, default=None, alias_priority=1,  
                  validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
    'filter':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='filter',  
                  serialization_alias='filter'),  
    'in_scope':  
        FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
                  alias_priority=1, validation_alias='inScope',  
                  serialization_alias='in_scope'),  
    'index_language':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='indexLanguage',  
                  serialization_alias='index_language'),  
    'index_name':  
        FieldInfo(annotation=str, required=True, alias_priority=1,  
                  validation_alias='indexName', serialization_alias='index_name'),  
    'query_type':  
        FieldInfo(annotation=Literal['simple', 'semantic',  
                                    'vector', 'vectorSimpleHybrid', 'vectorSemanticHybrid'], required=False,  
                  default='simple', alias_priority=1, validation_alias='queryType',  
                  serialization_alias='query_type', metadata=[AfterValidator(func=<function  
to_snake>)]),  
    'role_information':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None, alias_priority=1,  
                  validation_alias='roleInformation',  
                  serialization_alias='role_information'),  
    'semantic_configuration':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                  alias_priority=1, validation_alias='semanticConfiguration',  
                  serialization_alias='semantic_configuration'),  
    'strictness':  
        FieldInfo(annotation=int, required=False, default=3, alias_priority=1,
```

```
validation_alias='strictness', serialization_alias='strictness'),
'top_n_documents': FieldInfo(annotation=Union[int, NoneType],
required=False, default=5, alias_priority=1,
validation_alias='topNDocuments', serialization_alias='top_n_documents'))
```

authentication

Python

```
authentication: ApiKeyAuthentication | None
```

endpoint

Python

```
endpoint: str | None
```

query_type

Python

```
query_type: Literal['simple', 'semantic', 'vector', 'vectorSimpleHybrid',
'vectorSemanticHybrid']
```

AzureChatCompletion Class

Reference

Azure Chat completion class.

Initialize an AzureChatCompletion service.

Inheritance [AzureOpenAIConfigBase](#) → [AzureChatCompletion](#)

[OpenAIChatCompletionBase](#) → [AzureChatCompletion](#)

[OpenAITextCompletionBase](#) → [AzureChatCompletion](#)

Constructor

Python

```
AzureChatCompletion(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
None} Required*	<xref:<xref:ad_token {str} > The service ID for the Azure deployment. (Optional)
None} Required*	The optional api key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional deployment. If provided, will override the value (chat_deployment_name) in the env vars or .env file.
None} Required*	The optional deployment endpoint. If provided will override the value in the env vars or .env file.
None} Required*	The optional deployment base_url. If provided will override the value in the env vars or .env file.
None}	

Name	Description
Required*	The optional deployment api version. If provided will override the value in the env vars or .env file.
None} Required*	The Azure Active Directory token. (Optional)
{AsyncAzureADTokenProvider} Required*	<xref:ad_token_provider> The Azure Active Directory token provider. (Optional)
{Mapping[str] Required*	str (<xref:default_headers> The default headers mapping of string keys to string values for HTTP requests. (Optional)
str]} Required*	The default headers mapping of string keys to string values for HTTP requests. (Optional)
use. Required*	<xref:<xref:async_client {AsyncAzureOpenAI}> <xref:None} -- An existing client to>>
vars. Required*	<xref:<xref:env_file_path {str}> <xref:None} -- Use the environment settings file as a fallback to using env>>
service_id	default value: None
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
ad_token	default value: None
ad_token_provider	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

<code>from_dict</code>	Initialize an Azure OpenAI service from a dictionary of settings.
<code>get_prompt_execution_settings_class</code>	Create a request settings object.
<code>split_message</code>	<p>Split a Azure On Your Data response into separate ChatMessageContents.</p> <p>If the message does not have three contents, and those three are one each of: FunctionCallContent, FunctionResultContent, and TextContent, it will not return three messages, potentially only one or two.</p> <p>The order of the returned messages is as expected by OpenAI.</p>

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureChatCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: service_id, and optionally: ad_auth, ad_token_provider, default_headers

get_prompt_execution_settings_class

Create a request settings object.

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

split_message

Split a Azure On Your Data response into separate ChatMessageContents.

If the message does not have three contents, and those three are one each of: FunctionCallContent, FunctionResultContent, and TextContent, it will not return three messages, potentially only one or two.

The order of the returned messages is as expected by OpenAI.

Python

```
static split_message(message: ChatMessageContent) ->
list[semantic_kernel.contents.chat_message_content.ChatMessageContent]
```

Parameters

[+] [Expand table](#)

Name	Description
message Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
            FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
                FieldInfo(annotation=int, required=False, default=0), 'service_id':  
                    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
                        FieldInfo(annotation=int, required=False, default=0)}
```

AzureChatPromptExecutionSettings Class

Reference

Specific settings for the Azure OpenAI Chat Completion endpoint.

Inheritance [OpenAIChatPromptExecutionSettings](#) →
AzureChatPromptExecutionSettings

Constructor

Python

```
AzureChatPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
response_format: str | None = None, tools: list[dict[str, Any]] | None =  
None, tool_choice: str | None = None, function_call: str | None = None,  
functions: list[dict[str, Any]] | None = None, messages: list[dict[str,  
Any]] | None = None, function_call_behavior: FunctionCallBehavior | None =  
None, extra_body: dict[str, Any] | ExtraBody | None = None)
```

Parameters

[+] Expand table

Name	Description
<code>service_id</code>	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>extension_data</code>	

Name	Description
Required*	
ai_model_id	
Required*	
frequency_penalty	
Required*	
logit_bias	
Required*	
max_tokens	
Required*	
number_of_responses	
Required*	
presence_penalty	
Required*	
seed	
Required*	
stop	
Required*	
stream	
Required*	
temperature	
Required*	
top_p	
Required*	
user	
Required*	
response_format	
Required*	
tools	
Required*	
tool_choice	
Required*	
function_call	
Required*	
functions	

Name	Description
Required*	
messages	
Required*	
function_call_behavior	
Required*	
extra_body	
Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=2, serialization_alias='model'), 'extension_data':  
    FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'extra_body': FieldInfo(annotation=Union[dict[str,  
    Any], ExtraBody, NoneType], required=False, default=None),  
    'frequency_penalty': FieldInfo(annotation=Union[float, NoneType],  
    required=False, default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]),  
    'function_call': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'function_call_behavior':  
    FieldInfo(annotation=Union[FunctionCallBehavior, NoneType],  
    required=False, default=None, exclude=True), 'functions':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None), 'logit_bias':  
    FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],  
    required=False, default=None), 'max_tokens':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
    metadata=[Gt(gt=0)]), 'messages':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None), 'number_of_responses':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
    alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),  
    Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,  
    NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),  
    Le(le=2.0)]), 'response_format': FieldInfo(annotation=Union[str,  
    NoneType], required=False, default=None), 'seed':  
    FieldInfo(annotation=Union[int, NoneType], required=False, default=None),  
    'service_id': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None, metadata=[MinLen(min_length=1)]), 'stop':  
    FieldInfo(annotation=Union[str, list[str], NoneType], required=False,  
    default=None), 'stream': FieldInfo(annotation=bool, required=False,  
    default=False), 'temperature': FieldInfo(annotation=Union[float,  
    NoneType], required=False, default=None, metadata=[Ge(ge=0.0),  
    Le(le=2.0)]), 'tool_choice': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'tools':  
    FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
    required=False, default=None, metadata=[MaxLen(max_length=64)]), 'top_p':  
    FieldInfo(annotation=Union[float, NoneType], required=False,  
    default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

extra_body

Python

```
extra_body: dict[str, Any] | ExtraBody | None
```

response_format

Python

```
response_format: str | None
```

AzureCosmosDBDataSource Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureCosmosDBDataSource

Constructor

Python

```
AzureCosmosDBDataSource(*, type: Literal['azure_cosmos_db'] =  
    'azure_cosmos_db', parameters: AzureCosmosDBDataSourceParameters,  
    **extra_data: Any)
```

Keyword-Only Parameters

[] [Expand table](#)

Name	Description
type	default value: azure_cosmos_db
parameters Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'parameters':  
    FieldInfo(annotation=AzureCosmosDBDataSourceParameters, required=True,  
    alias_priority=1, validation_alias='parameters',  
    serialization_alias='parameters'), 'type':  
    FieldInfo(annotation=Literal['azure_cosmos_db'], required=False,  
    default='azure_cosmos_db', alias_priority=1, validation_alias='type',  
    serialization_alias='type')}
```

parameters

Python

```
parameters: AzureCosmosDBDataSourceParameters
```

type

Python

```
type: Literal['azure_cosmos_db']
```

AzureCosmosDBDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureDataSourceParameters](#) → AzureCosmosDBDataSourceParameters

Constructor

Python

```
AzureCosmosDBDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, authentication: ConnectionStringAuthentication | None = None, databaseName: str | None = None, containerName: str | None = None, embeddingDependencyType: AzureEmbeddingDependency | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True

Name	Description
topNDocuments	default value: 5
semanticConfiguration Required*	
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	
authentication Required*	
databaseName Required*	
containerName Required*	
embeddingDependencyType Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'authentication':  
    FieldInfo(annotation=Union[ConnectionStringAuthentication, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='authentication', serialization_alias='authentication'),  
    'container_name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='containerName', serialization_alias='container_name'),  
    'database_name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='databaseName', serialization_alias='database_name'),  
    'embedding_dependency':  
        FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
        required=False, default=None, alias_priority=1,  
        validation_alias='embeddingDependency',  
        serialization_alias='embedding_dependency'), 'embedding_dependency_type':  
            FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
            required=False, default=None, alias_priority=1,  
            validation_alias='embeddingDependencyType',  
            serialization_alias='embedding_dependency_type'), 'fields_mapping':  
                FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
                required=False, default=None, alias_priority=1,  
                validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
                'filter': FieldInfo(annotation=Union[str, NoneType], required=False,  
                default=None, alias_priority=1, validation_alias='filter',  
                serialization_alias='filter'), 'in_scope':  
                    FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
                    alias_priority=1, validation_alias='inScope',  
                    serialization_alias='in_scope'), 'index_language':  
                        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
                        alias_priority=1, validation_alias='indexLanguage',  
                        serialization_alias='index_language'), 'index_name':  
                            FieldInfo(annotation=str, required=True, alias_priority=1,  
                            validation_alias='indexName', serialization_alias='index_name'),  
                            'role_information': FieldInfo(annotation=Union[str, NoneType],
```

```
required=False, default=None, alias_priority=1,
validation_alias='roleInformation',
serialization_alias='role_information'), 'semantic_configuration':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=1, validation_alias='semanticConfiguration',
serialization_alias='semantic_configuration'), 'strictness':
FieldInfo(annotation=int, required=False, default=3, alias_priority=1,
validation_alias='strictness', serialization_alias='strictness'),
'top_n_documents': FieldInfo(annotation=Union[int, NoneType],
required=False, default=5, alias_priority=1,
validation_alias='topNDocuments', serialization_alias='top_n_documents')})
```

authentication

Python

```
authentication: ConnectionStringAuthentication | None
```

container_name

Python

```
container_name: str | None
```

database_name

Python

```
database_name: str | None
```

embedding_dependency_type

Python

```
embedding_dependency_type: AzureEmbeddingDependency | None
```

AzureDataSourceParameters Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureDataSourceParameters

Constructor

Python

```
AzureDataSourceParameters(*, indexName: str, indexLanguage: str | None = None, fieldsMapping: DataSourceFieldsMapping | None = None, inScope: bool | None = True, topNDocuments: int | None = 5, semanticConfiguration: str | None = None, roleInformation: str | None = None, filter: str | None = None, strictness: int = 3, embeddingDependency: AzureEmbeddingDependency | None = None, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
indexName Required*	
indexLanguage Required*	
fieldsMapping Required*	
inScope	default value: True
topNDocuments	default value: 5
semanticConfiguration Required*	

Name	Description
roleInformation Required*	
filter Required*	
strictness	default value: 3
embeddingDependency Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator': AliasGenerator(alias=None, validation_alias=<function to_camel>, serialization_alias=<function to_snake>), 'arbitrary_types_allowed': True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'embedding_dependency':  
    FieldInfo(annotation=Union[AzureEmbeddingDependency, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='embeddingDependency',  
    serialization_alias='embedding_dependency'), 'fields_mapping':  
    FieldInfo(annotation=Union[DataSourceFieldsMapping, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='fieldsMapping', serialization_alias='fields_mapping'),  
    'filter': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None, alias_priority=1, validation_alias='filter',  
    serialization_alias='filter'), 'in_scope':  
    FieldInfo(annotation=Union[bool, NoneType], required=False, default=True,  
    alias_priority=1, validation_alias='inScope',  
    serialization_alias='in_scope'), 'index_language':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='indexLanguage',  
    serialization_alias='index_language'), 'index_name':  
    FieldInfo(annotation=str, required=True, alias_priority=1,  
    validation_alias='indexName', serialization_alias='index_name'),  
    'role_information': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None, alias_priority=1,  
    validation_alias='roleInformation',  
    serialization_alias='role_information'), 'semantic_configuration':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='semanticConfiguration',  
    serialization_alias='semantic_configuration'), 'strictness':  
    FieldInfo(annotation=int, required=False, default=3, alias_priority=1,  
    validation_alias='strictness', serialization_alias='strictness'),  
    'top_n_documents': FieldInfo(annotation=Union[int, NoneType],  
    required=False, default=5, alias_priority=1,  
    validation_alias='topNDocuments', serialization_alias='top_n_documents')})
```

embedding_dependency

Python

```
embedding_dependency: AzureEmbeddingDependency | None
```

fields_mapping

Python

```
fields_mapping: DataSourceFieldsMapping | None
```

filter

Python

```
filter: str | None
```

in_scope

Python

```
in_scope: bool | None
```

index_language

Python

```
index_language: str | None
```

index_name

Python

```
index_name: str
```

role_information

Python

```
role_information: str | None
```

semantic_configuration

Python

```
semantic_configuration: str | None
```

strictness

Python

```
strictness: int
```

top_n_documents

Python

```
top_n_documents: int | None
```

AzureEmbeddingDependency Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → AzureEmbeddingDependency

Constructor

Python

```
AzureEmbeddingDependency(*, type: Literal['DeploymentName',  
'deployment_name'] = 'deployment_name', deploymentName: str | None = None,  
**extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
type	default value: deployment_name
deploymentName Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'deployment_name':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='deploymentName',  
    serialization_alias='deployment_name'), 'type':  
    FieldInfo(annotation=Literal['DeploymentName', 'deployment_name'],  
    required=False, default='deployment_name', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

deployment_name

Python

```
deployment_name: str | None
```

type

Python

```
type: Literal['DeploymentName', 'deployment_name']
```

AzureTextCompletion Class

Reference

Azure Text Completion class.

Initialize an AzureTextCompletion service.

Inheritance [AzureOpenAIConfigBase](#) → [AzureTextCompletion](#)
[OpenAITextCompletionBase](#) → [AzureTextCompletion](#)

Constructor

Python

```
AzureTextCompletion(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
service_id	The service ID for the Azure deployment. (Optional) default value: None
None} Required*	<xref:<xref:api_version {str}> > The optional api key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file.
None} Required*	The optional deployment endpoint. If provided will override the value in the env vars or .env file.
None} Required*	The optional deployment base_url. If provided will override the value in the env vars or .env file.
None}	

Name	Description
Required*	The optional deployment api version. If provided will override the value in the env vars or .env file.
ad_token	The Azure Active Directory token. (Optional) default value: None
ad_token_provider	The Azure Active Directory token provider. (Optional) default value: None
default_headers	The default headers mapping of string keys to string values for HTTP requests. (Optional) default value: None
use. Required*	<xref:async_client {Optional}> [<xref:AsyncAzureOpenAI>] <xref:> -- An existing client to>
to Required*	<xref:><xref:env_file_path {str} <xref:None}> -- Use the environment settings file as a fallback>> environment variables. (Optional)
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Azure OpenAI service from a dictionary of settings.
------------------	---

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureTextCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: deployment_name, endpoint, api_key and optionally: api_version, ad_auth

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
        FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
        FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
        FieldInfo(annotation=int, required=False, default=0), 'service_id':  
        FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
        FieldInfo(annotation=int, required=False, default=0)}
```

AzureTextEmbedding Class

Reference

Azure Text Embedding class.

Note: This class is experimental and may change in the future.

Initialize an AzureTextEmbedding service.

service_id: The service ID. (Optional) api_key {str | None}: The optional api key. If provided, will override the value in the

env vars or .env file.

deployment_name {str | None}: The optional deployment. If provided, will override the value (text_deployment_name) in the env vars or .env file.

endpoint {str | None}: The optional deployment endpoint. If provided will override the value in the env vars or .env file.

base_url {str | None}: The optional deployment base_url. If provided will override the value in the env vars or .env file.

api_version {str | None}: The optional deployment api version. If provided will override the value in the env vars or .env file.

ad_token {str | None}: The Azure AD token for authentication. (Optional) ad_auth {AsyncAzureADTokenProvider | None}: Whether to use Azure Active Directory authentication.

(Optional) The default value is False.

default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

async_client {Optional[AsyncAzureOpenAI]} – An existing client to use. (Optional)
env_file_path {str | None} – Use the environment settings file as a fallback to

environment variables. (Optional)

Inheritance [AzureOpenAIConfigBase](#) → AzureTextEmbedding

[OpenAITextEmbeddingBase](#) → AzureTextEmbedding

Constructor

Python

```
AzureTextEmbedding(service_id: str | None = None, api_key: str | None = None, deployment_name: str | None = None, endpoint: str | None = None, base_url: str | None = None, api_version: str | None = None, ad_token: str | None = None, ad_token_provider: Callable[[], str] | Awaitable[str] | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncAzureOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None
api_key	default value: None
deployment_name	default value: None
endpoint	default value: None
base_url	default value: None
api_version	default value: None
ad_token	default value: None
ad_token_provider	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

from_dict	Initialize an Azure OpenAI service from a dictionary of settings.
-----------	---

from_dict

Initialize an Azure OpenAI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> AzureTextEmbedding
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service. should contains keys: deployment_name, endpoint, api_key and optionally: api_version, ad_auth

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,  
        default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':  
    FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
    FieldInfo(annotation=int, required=False, default=0), 'service_id':  
    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
    FieldInfo(annotation=int, required=False, default=0)}
```

is_experimental

Python

```
is_experimental = True
```

ConnectionStringAuthentication Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [AzureChatRequestBase](#) → `ConnectionStringAuthentication`

Constructor

Python

```
ConnectionStringAuthentication(*, type: Literal['ConnectionString',  
'connection_string'] = 'connection_string', connectionString: str | None =  
None, **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>type</code>	default value: <code>connection_string</code>
<code>connectionString</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'connection_string':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='connectionString',  
    serialization_alias='connection_string'), 'type':  
    FieldInfo(annotation=Literal['ConnectionString', 'connection_string'],  
    required=False, default='connection_string', alias_priority=1,  
    validation_alias='type', serialization_alias='type', metadata=  
    [AfterValidator(func=<function to_snake>)])}
```

connection_string

Python

```
connection_string: str | None
```

type

Python

```
type: Literal['ConnectionString', 'connection_string']
```

DataSourceFieldsMapping Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AzureChatRequestBase](#) → DataSourceFieldsMapping

Constructor

Python

```
DataSourceFieldsMapping(*, titleField: str | None = None, urlField: str |  
None = None, filepathField: str | None = None, contentFields: list[str] |  
None = None, vectorFields: list[str] | None = None, contentFieldsSeparator:  
str | None = '\n', **extra_data: Any)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
titleField Required*	
urlField Required*	
filepathField Required*	
contentFields Required*	
vectorFields Required*	
contentFieldsSeparator	default value:

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'alias_generator':  
    AliasGenerator(alias=None, validation_alias=<function to_camel>,  
    serialization_alias=<function to_snake>), 'arbitrary_types_allowed':  
    True, 'extra': 'allow', 'populate_by_name': True, 'use_enum_values':  
    True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content_fields':  
    FieldInfo(annotation=Union[list[str], NoneType], required=False,  
    default=None, alias_priority=1, validation_alias='contentFields',  
    serialization_alias='content_fields'), 'content_fields_separator':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default='\n',  
    alias_priority=1, validation_alias='contentFieldsSeparator',  
    serialization_alias='content_fields_separator'), 'filepath_field':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='filepathField',  
    serialization_alias='filepath_field'), 'title_field':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
    alias_priority=1, validation_alias='titleField',
```

```
    serialization_alias='title_field'), 'url_field':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        alias_priority=1, validation_alias='urlField',  
        serialization_alias='url_field'), 'vector_fields':  
        FieldInfo(annotation=Union[list[str], NoneType], required=False,  
        default=None, alias_priority=1, validation_alias='vectorFields',  
        serialization_alias='vector_fields'))}
```

content_fields

Python

```
content_fields: list[str] | None
```

content_fields_separator

Python

```
content_fields_separator: str | None
```

filepath_field

Python

```
filepath_field: str | None
```

title_field

Python

```
title_field: str | None
```

url_field

Python

```
url_field: str | None
```

vector_fields

Python

```
vector_fields: list[str] | None
```

ExtraBody Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → ExtraBody

Constructor

Python

```
ExtraBody(*, data_sources:  
list[typing.Annotated[typing.Union[semantic_kernel.connectors.ai.open_ai.pro  
mpt_execution_settings.azure_chat_prompt_execution_settings.AzureAISearc  
hData  
sSource,  
semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_chat_p  
rompt_execution_settings.AzureCosmosDBDataSource],  
FieldInfo(annotation=NoneType, required=True, discriminator='type')]] | None  
= None, input_language: str | None = None, output_language: str | None =  
None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
data_sources Required*	
input_language Required*	
output_language Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'data_sources':
FieldInfo(annotation=Union[list[Annotated[Union[AzureAIDataSource,
AzureCosmosDBDataSource], FieldInfo(annotation=NoneType, required=True,
discriminator='type')]], NoneType], required=False, default=None),
'input_language': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None, alias_priority=2,
serialization_alias='inputLanguage'), 'output_language':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='outputLanguage')}
```

data_sources

Python

```
data_sources:
list[typing.Annotated[typing.Union[semantic_kernel.connectors.ai.open_ai.
```

```
prompt_execution_settings.azure_chat_prompt_execution_settings.AzureAISe
rchDataSource,
semantic_kernel.connectors.ai.open_ai.prompt_execution_settings.azure_cha
t_prompt_execution_settings.AzureCosmosDBDataSource],
FieldInfo(annotation=NoneType, required=True, discriminator='type')]] | |
None
```

input_language

Python

```
input_language: str | None
```

output_language

Python

```
output_language: str | None
```

OpenAIChatCompletion Class

Reference

OpenAI Chat completion class.

Initialize an OpenAIChatCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

Inheritance [OpenAIConfigBase](#) → OpenAIChatCompletion

[OpenAIChatCompletionBase](#) → OpenAIChatCompletion

[OpenAITextCompletionBase](#) → OpenAIChatCompletion

Constructor

Python

```
OpenAIChatCompletion(ai_model_id: str | None = None, service_id: str | None = None, api_key: str | None = None, org_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] Expand table

Name	Description
name Required*	<xref:<xref:ai_model_id {str} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>
override Required*	<xref:<xref:api_key {str} <xref:None} -- The optional API key to use. If provided will>>

Name	Description
	the env vars or .env file value.
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
fallback Required*	<xref:><xref:env_file_path {str}> <xref:None> -- Use the environment settings file as a>> to environment variables. (Optional)
ai_model_id	default value: None
service_id	default value: None
api_key	default value: None
org_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Open AI service from a dictionary of settings.
---------------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

<code>from_dict(settings: dict[str, str]) -> OpenAIChatCompletion</code>

Parameters

[\[\] Expand table](#)

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':
```

```
    FieldInfo(annotation=int, required=False, default=0), 'service_id':  
    FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
    FieldInfo(annotation=int, required=False, default=0)})
```

OpenAIChatPromptExecutionSettings Class

Reference

Specific settings for the Chat Completion endpoint.

Inheritance [OpenAIPromptExecutionSettings](#) → OpenAIChatPromptExecutionSettings

Constructor

Python

```
OpenAIChatPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
response_format: dict[Literal['type'], Literal['text', 'json_object']] |  
None = None, tools: list[dict[str, Any]] | None = None, tool_choice: str |  
None = None, function_call: str | None = None, functions: list[dict[str,  
Any]] | None = None, messages: list[dict[str, Any]] | None = None,  
function_call_behavior: FunctionCallBehavior | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
extension_data Required*	

Name	Description
ai_model_id Required*	
frequency_penalty Required*	
logit_bias Required*	
max_tokens Required*	
number_of_responses Required*	
presence_penalty Required*	
seed Required*	
stop Required*	
stream Required*	
temperature Required*	
top_p Required*	
user Required*	
response_format Required*	
tools Required*	
tool_choice Required*	
function_call Required*	
functions Required*	

Name	Description
messages Required*	
function_call_behavior Required*	

Methods

[+] [Expand table](#)

[validate_function_call](#)

validate_function_call

Python

```
validate_function_call(v: str | list[dict[str, Any]] | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
v	default value: None

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
alias_priority=2, serialization_alias='model'), 'extension_data':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'frequency_penalty':  
FieldInfo(annotation=Union[float, NoneType], required=False,  
default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]), 'function_call':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'function_call_behavior':  
FieldInfo(annotation=Union[FunctionCallBehavior, NoneType],  
required=False, default=None, exclude=True), 'functions':  
FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
required=False, default=None), 'logit_bias':  
FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],  
required=False, default=None), 'max_tokens':  
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
metadata=[Gt(gt=0)]), 'messages':  
FieldInfo(annotation=Union[list[dict[str, Any]], NoneType],  
required=False, default=None), 'number_of_responses':  
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),  
Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,  
NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),  
Le(le=2.0)]), 'response_format':  
FieldInfo(annotation=Union[dict[Literal['type'], Literal['text',  
'json_object']], NoneType], required=False, default=None), 'seed':  
FieldInfo(annotation=Union[int, NoneType], required=False, default=None),  
'service_id': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=None, metadata=[MinLen(min_length=1)]), 'stop':  
FieldInfo(annotation=Union[str, list[str], NoneType], required=False,  
default=None), 'stream': FieldInfo(annotation=bool, required=False),
```

```
default=False), 'temperature': FieldInfo(annotation=Union[float, NoneType], required=False, default=None, metadata=[Ge(ge=0.0), Le(le=2.0)]), 'tool_choice': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'tools': FieldInfo(annotation=Union[list[dict[str, Any]], NoneType], required=False, default=None, metadata=[MaxLen(max_length=64)]), 'top_p': FieldInfo(annotation=Union[float, NoneType], required=False, default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user': FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

function_call

Python

```
function_call: str | None
```

function_call_behavior

Python

```
function_call_behavior: FunctionCallBehavior | None
```

functions

Python

```
functions: list[dict[str, Any]] | None
```

messages

Python

```
messages: list[dict[str, Any]] | None
```

response_format

Python

```
response_format: dict[Literal['type'], Literal['text', 'json_object']] | None
```

tool_choice

Python

```
tool_choice: str | None
```

tools

Python

```
tools: list[dict[str, Any]] | None
```

OpenAIPromptExecutionSettings Class

Reference

Common request settings for (Azure) OpenAI services.

Inheritance [PromptExecutionSettings](#) → OpenAIPromptExecutionSettings

Constructor

Python

```
OpenAIPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None)
```

Parameters

[+] Expand table

Name	Description
service_id	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
extension_data	
Required*	
ai_model_id	
Required*	
frequency_penalty	
Required*	
logit_bias	

Name	Description
Required*	
max_tokens	
Required*	
number_of_responses	
Required*	
presence_penalty	
Required*	
seed	
Required*	
stop	
Required*	
stream	
Required*	
temperature	
Required*	
top_p	
Required*	
user	
Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='model'), 'extension_data':
FieldInfo(annotation=dict[str, Any], required=False,
default_factory=dict), 'frequency_penalty':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]), 'logit_bias':
FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],
required=False, default=None), 'max_tokens':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
metadata=[Gt(gt=0)]), 'number_of_responses':
FieldInfo(annotation=Union[int, NoneType], required=False, default=None,
alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),
Le(le=128)]), 'presence_penalty': FieldInfo(annotation=Union[float,
NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),
Le(le=2.0)]), 'seed': FieldInfo(annotation=Union[int, NoneType],
required=False, default=None), 'service_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None,
metadata=[MinLen(min_length=1)]), 'stop': FieldInfo(annotation=Union[str,
list[str], NoneType], required=False, default=None), 'stream':
FieldInfo(annotation=bool, required=False, default=False), 'temperature':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=0.0), Le(le=2.0)]), 'top_p':
FieldInfo(annotation=Union[float, NoneType], required=False,
default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]), 'user':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

frequency_penalty

Python

```
frequency_penalty: float | None
```

logit_bias

Python

```
logit_bias: dict[str | int, float] | None
```

max_tokens

Python

```
max_tokens: int | None
```

number_of_responses

Python

```
number_of_responses: int | None
```

presence_penalty

Python

```
presence_penalty: float | None
```

seed

Python

```
seed: int | None
```

stop

Python

```
stop: str | list[str] | None
```

stream

Python

```
stream: bool
```

temperature

Python

```
temperature: float | None
```

top_p

Python

```
top_p: float | None
```

user

Python

```
user: str | None
```

OpenAITextCompletion Class

Reference

OpenAI Text Completion class.

Initialize an OpenAITextCompletion service.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers: The default headers mapping of string keys to string values for HTTP requests. (Optional)

Inheritance [OpenAITextCompletionBase](#) → OpenAITextCompletion
[OpenAIConfigBase](#) → OpenAITextCompletion

Constructor

Python

```
OpenAITextCompletion(ai_model_id: str | None = None, api_key: str | None = None, org_id: str | None = None, service_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] [Expand table](#)

Name	Description
name Required*	<xref:<xref:ai_model_id {str} <xref:None} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>
override Required*	<xref:<xref:api_key {str} <xref:None} -- The optional API key to use. If provided will>> the env vars or .env file value.

Name	Description
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
to Required*	<xref:><xref:env_file_path {str}> <xref:None> -- Use the environment settings file as a fallback>> environment variables. (Optional)
ai_model_id	default value: None
api_key	default value: None
org_id	default value: None
service_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[+] Expand table

from_dict	Initialize an Open AI service from a dictionary of settings.
------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> OpenAITextCompletion
```

Parameters

[+] Expand table

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':
FieldInfo(annotation=int, required=False, default=0), 'service_id':
FieldInfo(annotation=str, required=False, default=''), 'total_tokens':
FieldInfo(annotation=int, required=False, default=0)}
```

OpenAITextEmbedding Class

Reference

OpenAI Text Embedding class.

Note: This class is experimental and may change in the future.

Initializes a new instance of the OpenAITextCompletion class.

:param : the env vars or .env file value. :param org_id {str | None} – The optional org ID to use. If provided will override: the env vars or .env file value. :param : the env vars or .env file value. :param default_headers {Optional[Mapping[str: The default headers mapping of string keys to

string values for HTTP requests. (Optional)

Inheritance [OpenAIConfigBase](#) → OpenAITextEmbedding

[OpenAITextEmbeddingBase](#) → OpenAITextEmbedding

Constructor

Python

```
OpenAITextEmbedding(ai_model_id: str, api_key: str | None = None, org_id: str | None = None, service_id: str | None = None, default_headers: Mapping[str, str] | None = None, async_client: AsyncOpenAI | None = None, env_file_path: str | None = None)
```

Parameters

[] [Expand table](#)

Name	Description
name Required*	<xref:<xref:ai_model_id {str} -- OpenAI model>> https://platform.openai.com/docs/models ↗
see Required*	https://platform.openai.com/docs/models ↗
settings. Required*	<xref:<xref:service_id {str} <xref:None} -- Service ID tied to the execution>>

Name	Description
override Required*	<xref:><xref:api_key {str} <xref:None> -- The optional API key to use. If provided will>> the env vars or .env file value.
str]]} Required*	The default headers mapping of string keys to string values for HTTP requests. (Optional)
use. Required*	<xref:async_client {Optional}>[<xref:AsyncOpenAI>]<xref:> -- An existing client to>
as Required*	<xref:><xref:env_file_path {str} <xref:None> -- Use the environment settings file>> a fallback to environment variables. (Optional)
ai_model_id Required*	
api_key	default value: None
org_id	default value: None
service_id	default value: None
default_headers	default value: None
async_client	default value: None
env_file_path	default value: None

Methods

[\[\] Expand table](#)

from_dict	Initialize an Open AI service from a dictionary of settings.
---------------------------	--

from_dict

Initialize an Open AI service from a dictionary of settings.

Python

```
from_dict(settings: dict[str, str]) -> OpenAITextEmbedding
```

Parameters

Name	Description
settings Required*	A dictionary of settings for the service.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=str, required=True, metadata=
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,
strict=None, min_length=1, max_length=None, pattern=None)]),
'ai_model_type': FieldInfo(annotation=OpenAIModelTypes, required=False,
default=<OpenAIModelTypes.CHAT: 'chat'>), 'client':
```

```
FieldInfo(annotation=AsyncOpenAI, required=True), 'completion_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'prompt_tokens':  
FieldInfo(annotation=int, required=False, default=0), 'service_id':  
FieldInfo(annotation=str, required=False, default=''), 'total_tokens':  
FieldInfo(annotation=int, required=False, default=0)}
```

is_experimental

Python

```
is_experimental = True
```

OpenAITextPromptExecutionSettings Class

Reference

Specific settings for the completions endpoint.

Inheritance [OpenAIPromptExecutionSettings](#) → OpenAITextPromptExecutionSettings

Constructor

Python

```
OpenAITextPromptExecutionSettings(service_id: str | None = None, *,  
extension_data: dict[str, Any] = None, ai_model_id: str | None = None,  
frequency_penalty: float | None = None, logit_bias: dict[str | int, float] |  
None = None, max_tokens: int | None = None, number_of_responses: int | None  
= None, presence_penalty: float | None = None, seed: int | None = None,  
stop: str | list[str] | None = None, stream: bool = False, temperature:  
float | None = None, top_p: float | None = None, user: str | None = None,  
prompt: str | None = None, best_of: int | None = None, echo: bool = False,  
logprobs: int | None = None, suffix: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
service_id	default value: None

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
extension_data Required*	
ai_model_id Required*	

Name	Description
frequency_penalty Required*	
logit_bias Required*	
max_tokens Required*	
number_of_responses Required*	
presence_penalty Required*	
seed Required*	
stop Required*	
stream Required*	
temperature Required*	
top_p Required*	
user Required*	
prompt Required*	
best_of Required*	
echo Required*	
logprobs Required*	
suffix Required*	

Methods

[Expand table](#)

<code>check_best_of_and_n</code>	Check that the <code>best_of</code> parameter is not greater than the <code>number_of_responses</code> parameter.
----------------------------------	---

check_best_of_and_n

Check that the `best_of` parameter is not greater than the `number_of_responses` parameter.

Python	<code>check_best_of_and_n() -> OpenAITextPromptExecutionSettings</code>
--------	--

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python	<code>model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}</code>
--------	---

model_config

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][`pydantic.config.ConfigDict`].

Python	<code>model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}</code>
--------	--

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {  
    'ai_model_id':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        alias_priority=2, serialization_alias='model'),  
    'best_of':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Ge(ge=1)]),  
    'echo': FieldInfo(annotation=bool, required=False, default=False),  
    'extension_data': FieldInfo(annotation=dict[str, Any],  
        required=False, default_factory=dict),  
    'frequency_penalty':  
        FieldInfo(annotation=Union[float, NoneType], required=False,  
        default=None, metadata=[Ge(ge=-2.0), Le(le=2.0)]),  
    'logit_bias':  
        FieldInfo(annotation=Union[dict[Union[str, int], float], NoneType],  
        required=False, default=None),  
    'logprobs':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Ge(ge=0), Le(le=5)]),  
    'max_tokens':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        metadata=[Gt(gt=0)]),  
    'number_of_responses':  
        FieldInfo(annotation=Union[int, NoneType], required=False, default=None,  
        alias_priority=2, serialization_alias='n', metadata=[Ge(ge=1),  
        Le(le=128)]),  
    'presence_penalty': FieldInfo(annotation=Union[float,  
        NoneType], required=False, default=None, metadata=[Ge(ge=-2.0),  
        Le(le=2.0)]),  
    'prompt': FieldInfo(annotation=Union[str, NoneType],  
        required=False, default=None),  
    'seed': FieldInfo(annotation=Union[int,  
        NoneType], required=False, default=None),  
    'service_id':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None,  
        metadata=[MinLen(min_length=1)]),  
    'stop': FieldInfo(annotation=Union[str,  
        list[str], NoneType], required=False, default=None),  
    'stream':  
        FieldInfo(annotation=bool, required=False, default=False),  
    'suffix':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'temperature': FieldInfo(annotation=Union[float, NoneType],  
        required=False, default=None, metadata=[Ge(ge=0.0), Le(le=2.0)]),  
    'top_p': FieldInfo(annotation=Union[float, NoneType], required=False,  
        default=None, metadata=[Ge(ge=0.0), Le(le=1.0)]),  
    'user':  
        FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}  
}
```

best_of

Python

```
best_of: int | None
```

echo

Python

```
echo: bool
```

logprobs

Python

```
logprobs: int | None
```

prompt

Python

```
prompt: str | None
```

suffix

Python

```
suffix: str | None
```

chat_completion_client_base Module

Reference

Classes

 [Expand table](#)

<p>ChatCompletionClientBase</p>	<p>Create a new model by parsing and validating input data from keyword arguments.</p>
---	--

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

ChatCompletionClientBase Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [AIServiceClientBase](#) → ChatCompletionClientBase

[ABC](#) → ChatCompletionClientBase

Constructor

Python

```
ChatCompletionClientBase(*, ai_model_id: str, service_id: str = '')
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
ai_model_id Required*	
service_id Required*	

Methods

[\[\] Expand table](#)

get_chat_message_contents	This is the method that is called from the kernel to get a response from a chat-optimized LLM.
get_streaming_chat_message_contents	This is the method that is called from the kernel to get a stream response from a chat-optimized LLM.

get_chat_message_contents

This is the method that is called from the kernel to get a response from a chat-optimized LLM.

Python

```
abstract async get_chat_message_contents(chat_history: ChatHistory,  
settings: PromptExecutionSettings, **kwargs: Any) ->  
list['ChatMessageContent']
```

Parameters

[+] Expand table

Name	Description
object Required*	<xref:<xref:chat_history {ChatHistory} -- A list> of <xref:chats in a chat_history>> rendered into messages from system, user, assistant and tools.
be Required*	<xref:<xref:that can>> rendered into messages from system, user, assistant and tools.
request. Required*	<xref:<xref:settings {PromptExecutionSettings} -- Settings for the>>
{Dict[str] Required*	str (<xref:kwargs>
arguments. Required*	<code>Any</code>]<xref:-- The optional>
chat_history Required*	
settings Required*	

Returns

[+] Expand table

Type	Description
	Union[str, List[str]] – A string or list of strings representing the response(s) from the LLM.

get_streaming_chat_message_contents

This is the method that is called from the kernel to get a stream response from a chat-optimized LLM.

Python

```
abstract get_streaming_chat_message_contents(chat_history: ChatHistory,
settings: PromptExecutionSettings, **kwargs: Any) ->
AsyncGenerator[list['StreamingChatMessageContent'], Any]
```

Parameters

[Expand table](#)

Name	Description
chat_history Required*	<xref:chat_history {ChatHistory} -- A list> of <xref:chat> set of chat_history, from system, user, assistant and function.
a Required*	<xref:<xref:that can be rendered into>> set of chat_history, from system, user, assistant and function.
request. Required*	<xref:<xref:settings {PromptExecutionSettings} -- Settings for the>>
{Dict[str Required*	str (<xref:kwargs>
arguments. Required*	Any]<xref:} -- The optional>
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=str, required=True, metadata=  
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
strict=None, min_length=1, max_length=None, pattern=None)]),  
'service_id': FieldInfo(annotation=str, required=False, default='')}
```

embedding_generator_base Module

Reference

Classes

 Expand table

[EmbeddingGeneratorBase](#) Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

EmbeddingGeneratorBase Class

Reference

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [AIServiceClientBase](#) → EmbeddingGeneratorBase

[ABC](#) → EmbeddingGeneratorBase

Constructor

Python

```
EmbeddingGeneratorBase(*, ai_model_id: str, service_id: str = '')
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>ai_model_id</code> Required*	
<code>service_id</code> Required*	

Methods

[\[\] Expand table](#)

[generate_embeddings](#)

generate_embeddings

Python

```
abstract async generate_embeddings(texts: list[str], **kwargs: Any) ->  
ndarray
```

Parameters

[+] Expand table

Name	Description
texts Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'service_id': FieldInfo(annotation=str, required=False, default='')}
```

is_experimental

Python

```
is_experimental = True
```

function_call_behavior Module

Reference

Classes

 Expand table

EnabledFunctions	<p>Function call behavior for making a filtered set of functions available for tool calls.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
FunctionCallBehavior	<p>Class that controls function calling behavior.</p> <p>Properties: auto_invoke_kernel_functions: Check if the kernel functions should be auto-invoked. Determined as max_auto_invoke_attempts > 0.</p> <p>Class methods: AutoInvokeKernelFunctions: Returns KernelFunctions class with auto_invoke enabled, all functions. EnableKernelFunctions: Returns KernelFunctions class with auto_invoke disabled, all functions. EnableFunctions: Set the enable kernel functions flag, filtered functions, auto_invoke optional. RequiredFunction: Set the required function flag, auto_invoke optional.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
FunctionCallConfiguration	<p>Class that holds the configured functions for function calling.</p>
KernelFunctions	<p>Function call behavior for making all kernel functions available for tool calls.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p>

self is explicitly positional-only to allow *self* as a field name.

RequiredFunction

Function call behavior for making a single function available for tool calls.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

EnabledFunctions Class

Reference

Function call behavior for making a filtered set of functions available for tool calls.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [FunctionCallBehavior](#) → EnabledFunctions

Constructor

Python

```
EnabledFunctions(*, enable_kernel_functions: bool = True,  
max_auto_invoke_attempts: int = 5, filters: dict[Literal['excluded_plugins',  
'included_plugins', 'excluded_functions', 'included_functions'], list[str]])
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>enable_kernel_functions</code>	default value: True
<code>max_auto_invoke_attempts</code>	default value: 5
<code>filters</code> Required*	

Methods

[\[\] Expand table](#)

configure	Set the options for the tool call behavior in the settings.
---------------------------	---

configure

Set the options for the tool call behavior in the settings.

Python

```
configure(kernel: Kernel, update_settings_callback: Callable[..., None], settings: PromptExecutionSettings) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
kernel Required*	
update_settings_callback Required*	
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'enable_kernel_functions': FieldInfo(annotation=bool, required=False,  
default=True), 'filters':  
FieldInfo(annotation=dict[Literal['excluded_plugins', 'included_plugins',  
'excluded_functions', 'included_functions'], list[str]], required=True),  
'max_auto_invoke_attempts': FieldInfo(annotation=int, required=False,  
default=5)}
```

filters

Python

```
filters: dict[Literal['excluded_plugins', 'included_plugins',  
'excluded_functions', 'included_functions'], list[str]]
```

FunctionCallBehavior Class

Reference

Class that controls function calling behavior.

Properties: auto_invoke_kernel_functions: Check if the kernel functions should be auto-invoked. Determined as max_auto_invoke_attempts > 0.

Class methods: AutoInvokeKernelFunctions: Returns KernelFunctions class with auto_invoke enabled, all functions. EnableKernelFunctions: Returns KernelFunctions class with auto_invoke disabled, all functions. EnableFunctions: Set the enable kernel functions flag, filtered functions, auto_invoke optional. RequiredFunction: Set the required function flag, auto_invoke optional.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → FunctionCallBehavior

Constructor

Python

```
FunctionCallBehavior(*, enable_kernel_functions: bool = True,  
max_auto_invoke_attempts: int = 5)
```

Parameters

[+] Expand table

Name	Description
enable_kernel_functions Required*	bool Enable kernel functions.
max_auto_invoke_attempts Required*	int The maximum number of auto invoke attempts.

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>enable_kernel_functions</code>	default value: True
<code>max_auto_invoke_attempts</code>	default value: 5

Methods

[+] Expand table

<code>AutoInvokeKernelFunctions</code>	Returns KernelFunctions class with auto_invoke enabled.
<code>EnableFunctions</code>	Set the enable kernel functions flag.
<code>EnableKernelFunctions</code>	Returns KernelFunctions class with auto_invoke disabled. Function calls are enabled in this case, just not invoked.
<code>RequiredFunction</code>	Set the required function flag.
<code>configure</code>	Configures the settings for the function call behavior, the default version in this class, does nothing, use subclasses for different behaviors.

AutoInvokeKernelFunctions

Returns KernelFunctions class with auto_invoke enabled.

Python

```
AutoInvokeKernelFunctions() -> KernelFunctions
```

EnableFunctions

Set the enable kernel functions flag.

Python

```
EnableFunctions(auto_invoke: bool = False, *, filters:  
dict[Literal['excluded_plugins', 'included_plugins',
```

```
'excluded_functions', 'included_functions'], list[str]]) ->  
EnabledFunctions
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>auto_invoke</code>	default value: False

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>filters</code> Required*	

EnableKernelFunctions

Returns KernelFunctions class with auto_invoke disabled.

Function calls are enabled in this case, just not invoked.

Python

```
EnableKernelFunctions() -> KernelFunctions
```

RequiredFunction

Set the required function flag.

Python

```
RequiredFunction(auto_invoke: bool = False, *,  
function_fully_qualified_name: str) -> RequiredFunction
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>auto_invoke</code>	default value: False

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>function_fully_qualified_name</code> Required*	

configure

Configures the settings for the function call behavior, the default version in this class, does nothing, use subclasses for different behaviors.

Python

```
configure()
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>kernel</code> Required*	
<code>update_settings_callback</code> Required*	
<code>settings</code> Required*	

Attributes

auto_invoke_kernel_functions

Check if the kernel functions should be auto-invoked.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'enable_kernel_functions': FieldInfo(annotation=bool, required=False,  
default=True), 'max_auto_invoke_attempts': FieldInfo(annotation=int,  
required=False, default=5)}
```

enable_kernel_functions

Enable kernel functions.

Python

```
enable_kernel_functions: bool
```

max_auto_invoke_attempts

The maximum number of auto invoke attempts.

Python

```
max_auto_invoke_attempts: int
```

FunctionCallConfiguration Class

Reference

Class that holds the configured functions for function calling.

Inheritance builtins.object → FunctionCallConfiguration

Constructor

Python

```
FunctionCallConfiguration(available_functions:  
list['KernelFunctionMetadata'] | None = None, required_functions:  
list['KernelFunctionMetadata'] | None = None)
```

Parameters

 Expand table

Name	Description
available_functions	default value: None
required_functions	default value: None

Attributes

available_functions

Python

```
available_functions:  
list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMet  
adata] | None = None
```

required_functions

Python

```
required_functions:  
list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMet  
adata] | None = None
```

KernelFunctions Class

Reference

Function call behavior for making all kernel functions available for tool calls.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [FunctionCallBehavior](#) → KernelFunctions

Constructor

Python

```
KernelFunctions(*, enable_kernel_functions: bool = True,  
max_auto_invoke_attempts: int = 5)
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>enable_kernel_functions</code>	default value: True
<code>max_auto_invoke_attempts</code>	default value: 5

Methods

[\[\] Expand table](#)

configure	Set the options for the tool call behavior in the settings.
---------------------------	---

configure

Set the options for the tool call behavior in the settings.

Python

```
configure(kernel: Kernel, update_settings_callback: Callable[..., None], settings: PromptExecutionSettings) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	
update_settings_callback Required*	
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'enable_kernel_functions': FieldInfo(annotation=bool, required=False,  
default=True), 'max_auto_invoke_attempts': FieldInfo(annotation=int,  
required=False, default=5)}
```

RequiredFunction Class

Reference

Function call behavior for making a single function available for tool calls.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [FunctionCallBehavior](#) → RequiredFunction

Constructor

Python

```
RequiredFunction(*, enable_kernel_functions: bool = True,  
max_auto_invoke_attempts: int = 5, function_fully_qualified_name: str)
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>enable_kernel_functions</code>	default value: True
<code>max_auto_invoke_attempts</code>	default value: 5
<code>function_fully_qualified_name</code> Required*	

Methods

[\[\] Expand table](#)

configure	Set the options for the tool call behavior in the settings.
---------------------------	---

configure

Set the options for the tool call behavior in the settings.

Python

```
configure(kernel: Kernel, update_settings_callback: Callable[..., None], settings: PromptExecutionSettings) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
kernel Required*	
update_settings_callback Required*	
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'enable_kernel_functions': FieldInfo(annotation=bool, required=False,  
default=True), 'function_fully_qualified_name': FieldInfo(annotation=str,  
required=True), 'max_auto_invoke_attempts': FieldInfo(annotation=int,  
required=False, default=5)}
```

function_fully_qualified_name

Python

```
function_fully_qualified_name: str
```

prompt_execution_settings Module

Reference

Classes

 Expand table

[PromptExecutionSettings](#) Base class for prompt execution settings.

Can be used by itself or as a base class for other prompt execution settings. The methods are used to create specific prompt execution settings objects based on the keys in the extension_data field, this way you can create a generic PromptExecutionSettings object in your application, which get's mapped into the keys of the prompt execution settings that each services returns by using the service.get_prompt_execution_settings() method.

PromptExecutionSettings Class

Reference

Base class for prompt execution settings.

Can be used by itself or as a base class for other prompt execution settings. The methods are used to create specific prompt execution settings objects based on the keys in the extension_data field, this way you can create a generic PromptExecutionSettings object in your application, which get's mapped into the keys of the prompt execution settings that each services returns by using the service.get_prompt_execution_settings() method.

Inheritance [KernelBaseModel](#) → PromptExecutionSettings

Constructor

Python

```
PromptExecutionSettings(service_id: str | None = None, *, extension_data: dict[str, Any] = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
service_id	str The service ID to use for the request. default value: None
extension_data Required*	<xref:Dict>[str , <xref: any>],<xref:="" optional><br=""></xref:> Any additional data to send with the request. Defaults to None.
kwargs Required*	Any Additional keyword arguments, these are attempted to parse into the keys of the specific prompt execution settings.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>extension_data</code> Required*	

Methods

[\[\] Expand table](#)

<code>from_prompt_execution_settings</code>	Create a prompt execution settings from another prompt execution settings.
<code>pack_extension_data</code>	Update the extension data from the prompt execution settings.
<code>prepare_settings_dict</code>	Prepares the settings as a dictionary for sending to the AI service.
<code>unpack_extension_data</code>	Update the prompt execution settings from extension data. Does not overwrite existing values with None.
<code>update_from_prompt_execution_settings</code>	Update the keys from another prompt execution settings object.

`from_prompt_execution_settings`

Create a prompt execution settings from another prompt execution settings.

Python

```
from_prompt_execution_settings()
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>config</code> Required*	

pack_extension_data

Update the extension data from the prompt execution settings.

Python

```
pack_extension_data() -> None
```

prepare_settings_dict

Prepares the settings as a dictionary for sending to the AI service.

Python

```
prepare_settings_dict()
```

unpack_extension_data

Update the prompt execution settings from extension data.

Does not overwrite existing values with None.

Python

```
unpack_extension_data() -> None
```

update_from_prompt_execution_settings

Update the keys from another prompt execution settings object.

Python

```
update_from_prompt_execution_settings()
```

Parameters

 Expand table

Name	Description
config	

Name	Description
Required*	

Attributes

keys

Get the keys of the prompt execution settings.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'extension_data':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'service_id': FieldInfo(annotation=Union[str,
```

```
NoneType], required=False, default=None, metadata=[MinLen(min_length=1)])}
```

extension_data

Python

```
extension_data: dict[str, Any]
```

service_id

Python

```
service_id: str | None
```

text_completion_client_base Module

Reference

Classes

[+] Expand table

[TextCompletionClientBase](#) Base class for text completion AI services.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

TextCompletionClientBase Class

Reference

Base class for text completion AI services.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `AIServiceClientBase` → `TextCompletionClientBase`

`ABC` → `TextCompletionClientBase`

Constructor

Python

```
TextCompletionClientBase(*, ai_model_id: str, service_id: str = '')
```

Keyword-Only Parameters

[] Expand table

Name	Description
<code>ai_model_id</code> Required*	
<code>service_id</code> Required*	

Methods

[] Expand table

<code>get_streaming_text_contents</code>	This is the method that is called from the kernel to get a stream response from a text-optimized LLM.
--	---

get_text_contents

This is the method that is called from the kernel to get a response from a text-optimized LLM.

get_streaming_text_contents

This is the method that is called from the kernel to get a stream response from a text-optimized LLM.

Python

```
abstract get_streaming_text_contents(prompt: str, settings:  
PromptExecutionSettings) -> AsyncGenerator[list['StreamingTextContent'],  
Any]
```

Parameters

[+] Expand table

Name	Description
LLM. Required*	<xref:<xref:prompt {str} -- The prompt to send to the>>
request. Required*	<xref:<xref:settings {PromptExecutionSettings} -- Settings for the>>
prompt Required*	
settings Required*	

get_text_contents

This is the method that is called from the kernel to get a response from a text-optimized LLM.

Python

```
abstract async get_text_contents(prompt: str, settings:  
PromptExecutionSettings) -> list['TextContent']
```

Parameters

Name	Description
LLM. Required*	<xref:<xref:prompt {str} -- The prompt to send to the>>
request. Required*	<xref:<xref:settings {PromptExecutionSettings} -- Settings for the>>
Returns Required*	
response Required*	<code>list[TextContent]</code> <xref:-- A string> or <code>list</code> of <xref:strings representing the>
prompt Required*	
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=str, required=True, metadata=  
        [StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
            strict=None, min_length=1, max_length=None, pattern=None)]),  
    'service_id': FieldInfo(annotation=str, required=False, default='')}
```

PromptExecutionSettings Class

Reference

Base class for prompt execution settings.

Can be used by itself or as a base class for other prompt execution settings. The methods are used to create specific prompt execution settings objects based on the keys in the extension_data field, this way you can create a generic PromptExecutionSettings object in your application, which get's mapped into the keys of the prompt execution settings that each services returns by using the service.get_prompt_execution_settings() method.

Inheritance [KernelBaseModel](#) → PromptExecutionSettings

Constructor

Python

```
PromptExecutionSettings(service_id: str | None = None, *, extension_data: dict[str, Any] = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
service_id	str The service ID to use for the request. default value: None
extension_data Required*	<xref:Dict>[str , <xref: any>],<xref:="" optional><br=""></xref:> Any additional data to send with the request. Defaults to None.
kwargs Required*	Any Additional keyword arguments, these are attempted to parse into the keys of the specific prompt execution settings.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>extension_data</code> Required*	

Methods

[\[\] Expand table](#)

<code>from_prompt_execution_settings</code>	Create a prompt execution settings from another prompt execution settings.
<code>pack_extension_data</code>	Update the extension data from the prompt execution settings.
<code>prepare_settings_dict</code>	Prepares the settings as a dictionary for sending to the AI service.
<code>unpack_extension_data</code>	Update the prompt execution settings from extension data. Does not overwrite existing values with None.
<code>update_from_prompt_execution_settings</code>	Update the keys from another prompt execution settings object.

`from_prompt_execution_settings`

Create a prompt execution settings from another prompt execution settings.

Python

```
from_prompt_execution_settings()
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>config</code> Required*	

pack_extension_data

Update the extension data from the prompt execution settings.

Python

```
pack_extension_data() -> None
```

prepare_settings_dict

Prepares the settings as a dictionary for sending to the AI service.

Python

```
prepare_settings_dict()
```

unpack_extension_data

Update the prompt execution settings from extension data.

Does not overwrite existing values with None.

Python

```
unpack_extension_data() -> None
```

update_from_prompt_execution_settings

Update the keys from another prompt execution settings object.

Python

```
update_from_prompt_execution_settings()
```

Parameters

 Expand table

Name	Description
config	

Name	Description
Required*	

Attributes

keys

Get the keys of the prompt execution settings.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'extension_data':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'service_id': FieldInfo(annotation=Union[str,
```

```
NoneType], required=False, default=None, metadata=[MinLen(min_length=1)])}
```

extension_data

Python

```
extension_data: dict[str, Any]
```

service_id

Python

```
service_id: str | None
```

memory Package

Reference

Packages

[\[\] Expand table](#)

[chroma](#)

[milvus](#)

[qdrant](#)

[usearch](#)

Modules

[\[\] Expand table](#)

[memory_settings_base](#)

chroma Package

Reference

Modules

[] Expand table

[chroma_memory_store](#)

[utils](#)

Classes

[] Expand table

[ChromaMemoryStore](#) Note: This class is experimental and may change in the future.

ChromaMemoryStore provides an interface to store and retrieve data using ChromaDB. Collection names with uppercase characters are not supported by ChromaDB, they will be automatically converted.

chroma_memory_store Module

Reference

Classes

 Expand table

[ChromaMemoryStore](#) Note: This class is experimental and may change in the future.

ChromaMemoryStore provides an interface to store and retrieve data using ChromaDB. Collection names with uppercase characters are not supported by ChromaDB, they will be automatically converted.

ChromaMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

ChromaMemoryStore provides an interface to store and retrieve data using ChromaDB. Collection names with uppercase characters are not supported by ChromaDB, they will be automatically converted.

Inheritance [MemoryStoreBase](#) → ChromaMemoryStore

Constructor

Python

```
ChromaMemoryStore(persist_directory: str | None = None, client_settings:  
chromadb.config.Settings | None = None, **kwargs)
```

Parameters

[Expand table](#)

Name	Description
persist_directory	<xref:Optional>[str],<xref: optional> Path to the directory where data will be persisted. Defaults to None, which means the default settings for ChromaDB will be used. default value: None
client_settings	<xref:Optional>[<xref:"chromadb.config.Settings">],<xref: optional> A Settings instance to configure the ChromaDB client. Defaults to None, which means the default settings for ChromaDB will be used. default value: None
similarity_fetch_limit Required*	<xref:> <xref:semantic_kernel.connectors.memory.chroma.chroma_memory_store.int,> optional>> The maximum number of results to calculate cosine-similarity.

Examples

Create a ChromaMemoryStore with a local specified directory for data persistence

```
chroma_local_data_store =  
ChromaMemoryStore(persist_directory='/path/to/persist/directory')
```

Create a ChromaMemoryStore with a custom Settings instance

```
chroma_remote_data_store = ChromaMemoryStore(  
  
client_settings=Settings(chroma_api_impl="rest", chroma_server_host="xxx.xxx.xxx.xxx",  
chroma_server_http_port="8000"  
  
)  
  
)
```

Methods

 Expand table

create_collection	Creates a new collection in Chroma if it does not exist. To prevent downloading model file from embedding_function, embedding_function is set to "DoNotUseChromaEmbeddingFunction".
delete_collection	Deletes a collection.
does_collection_exist	Checks if a collection exists.
get	Gets a record.
get_batch	Gets a batch of records.
get_collection	
get_collections	Gets the list of collections.
get_nearest_match	Gets the nearest match to an embedding using cosine similarity.
get_nearest_matches	Gets the nearest matches to an embedding using cosine similarity.
remove	Removes a record.

remove_batch	Removes a batch of records.
upsert	Upserts a single MemoryRecord.
upsert_batch	Upserts a batch of records.

create_collection

Creates a new collection in Chroma if it does not exist. To prevent downloading model file from embedding_function, embedding_function is set to "DoNotUseChromaEmbeddingFunction".

Python

```
async create_collection(collection_name: str) -> None
```

Parameters

[\[+\]](#) Expand table

Name	Description
create. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
case. Required*	<xref:<xref:The name> of <xref:the collection will be converted to snake>>
collection_name Required*	

Returns

[\[+\]](#) Expand table

Type	Description
	None

delete_collection

Deletes a collection.

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
delete. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[+] Expand table

Type	Description
	None

does_collection_exist

Checks if a collection exists.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[+] Expand table

Name	Description
check. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[] Expand table

Type	Description
	bool – True if the collection exists; otherwise, False.

get

Gets a record.

Python

```
async get(collection_name: str, key: str, with_embedding: bool) ->
MemoryRecord
```

Parameters

[] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the record>>
record. Required*	<xref:key {str} -- The unique database key> of <xref:the>
collection_name Required*	
key Required*	
with_embedding Required*	

Returns

[] Expand table

Type	Description
	MemoryRecord – The record.

get_batch

Gets a batch of records.

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the records>>
records. Required*	<xref:keys {List}>[str]<xref:> -- The unique database keys> of <xref:the>
collection_name Required*	
keys Required*	
with_embeddings Required*	

Returns

[+] Expand table

Type	Description
	List[MemoryRecord] – The records.

get_collection

Python

```
async get_collection(collection_name: str) -> Collection | None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

get_collections

Gets the list of collections.

Python

```
async get_collections() -> list[str]
```

Returns

[+] Expand table

Type	Description
	List[str] – The list of collections.

get_nearest_match

Gets the nearest match to an embedding using cosine similarity.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float = 0.0, with_embedding: bool = True) ->  
tuple[semantic_kernel.memory.MemoryRecord, float]
```

Parameters

[+] Expand table

Name	Description
from.	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the nearest match>>

Name	Description
Required*	
to. Required*	<xref:<xref:embedding {ndarray} -- The embedding to find the nearest match>>
collection_name Required*	
embedding Required*	
min_relevance_score	default value: 0.0
with_embedding	default value: True

Returns

[] Expand table

Type	Description
	Tuple[MemoryRecord, float] – The record and the relevance score.

get_nearest_matches

Gets the nearest matches to an embedding using cosine similarity.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray, limit: int, min_relevance_score: float = 0.0, with_embeddings: bool = True) -> list[tuple[semantic_kernel.memory.MemoryRecord, float]]
```

Parameters

[] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the nearest matches>>
to. Required*	<xref:<xref:embedding {ndarray} -- The embedding to find the nearest matches>>

Name	Description
return. Required*	<xref:<xref:limit {int} -- The maximum number> of <xref:matches to>>
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score	default value: 0.0
with_embeddings	default value: True

Returns

[] Expand table

Type	Description
	List[Tuple[MemoryRecord, float]] – The records and their relevance scores.

remove

Removes a record.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to remove the record>>
remove. Required*	<xref:<xref:key {str} -- The unique database key> of <xref:the record to>>

Name	Description
collection_name Required*	
key Required*	

Returns

[+] Expand table

Type	Description
	None

remove_batch

Removes a batch of records.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to remove the records>>
remove. Required*	<xref:keys {List}>[str]<xref:-- The unique database keys> of <xref:the records to>
collection_name Required*	
keys Required*	

Returns

[Expand table](#)

Type	Description
	None

upsert

Upserts a single MemoryRecord.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[Expand table](#)

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the record>>
upsert. Required*	<xref:<xref:records {MemoryRecord} -- The record to>>
collection_name Required*	
record Required*	

Returns

[Expand table](#)

Type	Description
	List[str] – The unique database key of the record.

upsert_batch

Upserts a batch of records.

Python

```
async upsert_batch(collection_name: str, records:  
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[+] Expand table

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the records>>
upsert. Required*	<xref:records {List>[MemoryRecord]<xref:} -- The records to>
collection_name Required*	
records Required*	

Returns

[+] Expand table

Type	Description
	List[str] – The unique database keys of the records. In Pinecone, these are the record IDs.

Attributes

is_experimental

Python

```
is_experimental = True
```

utils Module

Reference

Functions

camel_to_snake

Python

```
camel_to_snake(camel_str)
```

Parameters

[\[\] Expand table](#)

Name	Description
camel_str Required*	

chroma_compute_similarity_scores

Computes the cosine similarity scores between a query embedding and a group of embeddings.

:param embedding {ndarray} – The query embedding.

:param embedding_array {ndarray} – The group of embeddings.

Python

```
chroma_compute_similarity_scores(embedding: ndarray, embedding_array:  
ndarray, **kwargs) -> ndarray
```

Parameters

[\[\] Expand table](#)

Name	Description
embedding Required*	

Name	Description
embedding_array Required*	

Returns

[\[\] Expand table](#)

Type	Description
	ndarray – The cosine similarity scores.

query_results_to_records

Python

```
query_results_to_records(results: QueryResult, with_embedding: bool) ->
list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[\[\] Expand table](#)

Name	Description
results Required*	
with_embedding Required*	

ChromaMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

ChromaMemoryStore provides an interface to store and retrieve data using ChromaDB. Collection names with uppercase characters are not supported by ChromaDB, they will be automatically converted.

Inheritance [MemoryStoreBase](#) → ChromaMemoryStore

Constructor

Python

```
ChromaMemoryStore(persist_directory: str | None = None, client_settings:  
chromadb.config.Settings | None = None, **kwargs)
```

Parameters

[] [Expand table](#)

Name	Description
persist_directory	<xref:Optional>[str],<xref: optional> Path to the directory where data will be persisted. Defaults to None, which means the default settings for ChromaDB will be used. default value: None
client_settings	<xref:Optional>[<xref:"chromadb.config.Settings">],<xref: optional> A Settings instance to configure the ChromaDB client. Defaults to None, which means the default settings for ChromaDB will be used. default value: None
similarity_fetch_limit Required*	<xref:<xref:semantic_kernel.connectors.memory.chroma.int, optional>> The maximum number of results to calculate cosine-similarity.

Examples

Create a ChromaMemoryStore with a local specified directory for data persistence

```
chroma_local_data_store =  
ChromaMemoryStore(persist_directory='/path/to/persist/directory')
```

Create a ChromaMemoryStore with a custom Settings instance

```
chroma_remote_data_store = ChromaMemoryStore(  
  
client_settings=Settings(chroma_api_impl="rest", chroma_server_host="xxx.xxx.xxx.xxx",  
chroma_server_http_port="8000"  
  
)  
)
```

Methods

[] [Expand table](#)

create_collection	Creates a new collection in Chroma if it does not exist. To prevent downloading model file from embedding_function, embedding_function is set to "DoNotUseChromaEmbeddingFunction".
delete_collection	Deletes a collection.
does_collection_exist	Checks if a collection exists.
get	Gets a record.
get_batch	Gets a batch of records.
get_collection	
get_collections	Gets the list of collections.
get_nearest_match	Gets the nearest match to an embedding using cosine similarity.
get_nearest_matches	Gets the nearest matches to an embedding using cosine similarity.
remove	Removes a record.

remove_batch	Removes a batch of records.
upsert	Upserts a single MemoryRecord.
upsert_batch	Upserts a batch of records.

create_collection

Creates a new collection in Chroma if it does not exist. To prevent downloading model file from embedding_function, embedding_function is set to "DoNotUseChromaEmbeddingFunction".

Python

```
async create_collection(collection_name: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
create. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
case. Required*	<xref:<xref:The name> of <xref:the collection will be converted to snake>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

delete_collection

Deletes a collection.

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
delete. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[+] Expand table

Type	Description
	None

does_collection_exist

Checks if a collection exists.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[+] Expand table

Name	Description
check. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name	

Name	Description
Required*	

Returns

[+] Expand table

Type	Description
	bool – True if the collection exists; otherwise, False.

get

Gets a record.

Python

```
async get(collection_name: str, key: str, with_embedding: bool) ->
MemoryRecord
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the record>>
record. Required*	<xref:key {str} -- The unique database key> of <xref:the>
collection_name Required*	
key Required*	
with_embedding Required*	

Returns

[+] Expand table

Type	Description
	MemoryRecord – The record.

get_batch

Gets a batch of records.

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the records>>
records. Required*	<xref:keys {List}>[str]<xref:> -- The unique database keys> of <xref:the>
collection_name Required*	
keys Required*	
with_embeddings Required*	

Returns

[+] Expand table

Type	Description
	List[MemoryRecord] – The records.

get_collection

Python

```
async get_collection(collection_name: str) -> Collection | None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

get_collections

Gets the list of collections.

Python

```
async get_collections() -> list[str]
```

Returns

[+] Expand table

Type	Description
	List[str] – The list of collections.

get_nearest_match

Gets the nearest match to an embedding using cosine similarity.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float = 0.0, with_embedding: bool = True) ->  
tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the nearest match>>
to. Required*	<xref:<xref:embedding {ndarray} -- The embedding to find the nearest match>>
collection_name Required*	
embedding Required*	
min_relevance_score	default value: 0.0
with_embedding	default value: True

Returns

[+] Expand table

Type	Description
	Tuple[MemoryRecord, float] – The record and the relevance score.

get_nearest_matches

Gets the nearest matches to an embedding using cosine similarity.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,
limit: int, min_relevance_score: float = 0.0, with_embeddings: bool =
True) -> list[tuple[semantic_kernel.memory.memory_record.MemoryRecord,
float]]
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to get the nearest matches>>
to. Required*	<xref:<xref:embedding {ndarray} -- The embedding to find the nearest matches>>
return. Required*	<xref:<xref:limit {int} -- The maximum number> of <xref:matches to>>
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score	default value: 0.0
with_embeddings	default value: True

Returns

[+] Expand table

Type	Description
	List[Tuple[MemoryRecord, float]] – The records and their relevance scores.

remove

Removes a record.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to remove the record>>
remove. Required*	<xref:<xref:key {str} -- The unique database key> of <xref:the record to>>
collection_name Required*	
key Required*	

Returns

[+] Expand table

Type	Description
	None

remove_batch

Removes a batch of records.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[+] Expand table

Name	Description
from. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to remove the records>>
remove. Required*	<xref:keys {List}>[str]<xref:> -- The unique database keys> of <xref:the records to>
collection_name Required*	

Name	Description
keys Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

upsert

Upserts a single MemoryRecord.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the record>>
upsert. Required*	<xref:<xref:records {MemoryRecord} -- The record to>>
collection_name Required*	
record Required*	

Returns

[\[\] Expand table](#)

Type	Description
	List[str] – The unique database key of the record.

upsert_batch

Upserts a batch of records.

Python

```
async upsert_batch(collection_name: str, records:
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[+] Expand table

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the records>>
upsert. Required*	<xref:records {List>}[MemoryRecord]<xref:> -- The records to>
collection_name Required*	
records Required*	

Returns

[+] Expand table

Type	Description
	List[str] – The unique database keys of the records. In Pinecone, these are the record IDs.

Attributes

is_experimental

Python

```
is_experimental = True
```

milvus Package

Reference

Modules

[] [Expand table](#)

[milvus_memory_store](#)

Classes

[] [Expand table](#)

[MilvusMemoryStore](#) Note: This class is experimental and may change in the future.

MilvusMemoryStore allows for searching for records using Milvus/Zilliz Cloud.

For more details on how to get the service started, take a look here: Milvus:
https://milvus.io/docs/get_started.md ↗ Zilliz Cloud:
<https://docs.zilliz.com/docs/quick-start> ↗

milvus_memory_store Module

Reference

Classes

[+] Expand table

`MilvusMemoryStore` Note: This class is experimental and may change in the future.

MilvusMemoryStore allows for searching for records using Milvus/Zilliz Cloud.

For more details on how to get the service started, take a look here: Milvus:
https://milvus.io/docs/get_started.md Zilliz Cloud:
<https://docs.zilliz.com/docs/quick-start>

Functions

`create_fields`

Note: This function is experimental and may change in the future.

Python

```
create_fields(dimensions: int) -> list[pymilvus.orm.schema.FieldSchema]
```

Parameters

[+] Expand table

Name	Description
<code>dimensions</code> Required*	

`memoryrecord_to_milvus_dict`

Convert a memoryrecord into a dict. Args: mem (MemoryRecord): MemoryRecord to convert.

Returns: dict: Dict result.

Note: This function is experimental and may change in the future.

Python

```
memoryrecord_to_milvus_dict(mem: MemoryRecord) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
mem Required*	

milvus_dict_to_memoryrecord

Convert Milvus search result dict into MemoryRecord.

Args: milvus_dict (dict): Search hit

Returns: MemoryRecord

Note: This function is experimental and may change in the future.

Python

```
milvus_dict_to_memoryrecord(milvus_dict: dict[str, Any]) -> MemoryRecord
```

Parameters

[\[\] Expand table](#)

Name	Description
milvus_dict Required*	

MilvusMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

MilvusMemoryStore allows for searching for records using Milvus/Zilliz Cloud.

For more details on how to get the service started, take a look here: Milvus:

https://milvus.io/docs/get_started.md ↗ Zilliz Cloud: <https://docs.zilliz.com/docs/quick-start> ↗

Inheritance [MemoryStoreBase](#) → MilvusMemoryStore

Constructor

Python

```
MilvusMemoryStore(uri: str = 'http://localhost:19530', token: str | None = None,  
**kwargs)
```

Parameters

[Expand table](#)

Name	Description
uri	<xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.str, optional>> The uri of the cluster. Defaults to " http://localhost:19530 ". default value: http://localhost:19530
token	<xref:Optional> [str],<xref: optional> The token to connect to the cluster if authentication is required. Defaults to None. default value: None

Methods

[Expand table](#)

create_collection	Create a Milvus collection.
delete_collection	Delete the specified collection. If all is True, all collections in the cluster will be removed.
does_collection_exist	Return if the collection exists in the cluster.

get	Get the MemoryRecord corresponding to the key.
get_batch	Get the MemoryRecords corresponding to the keys
get_collections	Return a list of present collections.
get_nearest_match	Find the nearest match for an embedding.
get_nearest_matches	Find the nearest <i>limit</i> matches for an embedding.
remove	Remove the specified record based on key.
remove_batch	Remove multiple records based on keys.
upsert	Upsert a single MemoryRecord into the collection.
upsert_batch	<i>summary</i>

create_collection

Create a Milvus collection.

Python

```
async create_collection(collection_name: str, dimension_num: int = 1536,
distance_type: str | None = 'IP', overwrite: bool = False, consistency: str =
'Session') -> None
```

Parameters

[\[+\]](#) [Expand table](#)

Name	Description
collection_name	<code>str</code> Required*
	The name of the collection.
dimension_num	<code><xref:Optional>[int],<xref: optional></code> The size of the embeddings being stored. Defaults to 1536. default value: 1536
distance_type	<code><xref:Optional>[str],<xref: optional></code> Which distance function, at the moment only "IP" and "L2" are supported. Defaults to "IP". default value: IP
overwrite	<code><xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.bool, optional>></code> Whether to overwrite any existing collection with the same name. Defaults to False.

Name	Description
	default value: False
consistency	<xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.str, optional>> Which consistency level to use: Strong, Session, Bounded, Eventually. Defaults to "Session". default value: Session

delete_collection

Delete the specified collection.

If all is True, all collections in the cluster will be removed.

Python

```
async delete_collection(collection_name: str | None = None, all: bool = False)
-> None
```

Parameters

[Expand table](#)

Name	Description
collection_name	<xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.str, optional>> The name of the collection to delete. Defaults to "". default value: None
all	<xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.bool, optional>> Whether to delete all collections. Defaults to False. default value: False

does_collection_exist

Return if the collection exists in the cluster.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str The name of the collection.

Returns

[Expand table](#)

Type	Description
bool	True if it exists, False otherwise.

get

Get the MemoryRecord corresponding to the key.

Python

```
async get(collection_name: str, key: str, with_embedding: bool) -> MemoryRecord
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str The collection to get from.
key Required*	str The ID to grab.
with_embedding Required*	bool Whether to include the embedding in the results.

Returns

[Expand table](#)

Type	Description
MemoryRecord	The MemoryRecord for the key.

get_batch

Get the MemoryRecords corresponding to the keys

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory_record.MemoryRecord]
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str <i>description</i>
keys Required*	<xref>List>[str] <i>description</i>
with_embeddings Required*	bool <i>description</i>

Returns

[Expand table](#)

Type	Description
List[MemoryRecord]	<i>description</i>

Exceptions

[Expand table](#)

Type	Description
Exception	<i>description</i>
e	<i>description</i>

get_collections

Return a list of present collections.

Python

```
async get_collections() -> list[str]
```

Returns

 Expand table

Type	Description
List[str]	List of collection names.

get_nearest_match

Find the nearest match for an embedding.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float = 0.0, with_embedding: bool = False) ->  
tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]
```

Parameters

 Expand table

Name	Description
collection_name Required*	str The collection to search.
embedding Required*	<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.ndarray> The embedding to search for.
min_relevance_score	<xref: <xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.float, optional>> 20. Defaults to 0.0. default value: 0.0
with_embedding	<xref: <xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.bool, optional>> Whether to include embedding in result. Defaults to False. default value: False

Returns

[+] [Expand table](#)

Type	Description
<code>Tuple[MemoryRecord, float]</code>	A tuple of record and distance.

get_nearest_matches

Find the nearest *limit* matches for an embedding.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray, limit: int,
min_relevance_score: float = 0.0, with_embeddings: bool = False) ->
list[tuple[semantic_kernel.memory.MemoryRecord, float]]
```

Parameters

[+] [Expand table](#)

Name	Description
collection_name Required*	str The collection to search.
embedding Required*	<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.ndarray> The embedding to search.
limit Required*	int The total results to display.
min_relevance_score	<xref: <xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.float, optional>> Minimum distance to include. Defaults to None. default value: 0.0
with_embeddings	<xref: <xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.bool, optional>> Whether to include embeddings in result. Defaults to False. default value: False

Returns

[Expand table](#)

Type	Description
List[Tuple[MemoryRecord, float]]	MemoryRecord and distance tuple.

Exceptions

[Expand table](#)

Type	Description
Exception	Missing collection
e	Failure to search

remove

Remove the specified record based on key.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name	str Required*
key	str Required*

remove_batch

Remove multiple records based on keys.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str Collection to remove from
keys Required*	<xref>List>[str] The list of keys.

Exceptions

[Expand table](#)

Type	Description
Exception	Collection doesnt exist.
e	Failure to remove key.

upsert

Upsert a single MemoryRecord into the collection.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str The name of the collection.
record Required*	<xref>semantic_kernel.connectors.memory.milvus.milvus_memory_store.MemoryRecord> The record to store.

Returns

[Expand table](#)

Type	Description
str	The ID of the inserted record.

upsert_batch

summary

Python

```
async upsert_batch(collection_name: str, records:
list[semantic_kernel.memory.memory_record.MemoryRecord], batch_size=100) ->
list[str]
```

Parameters

[] Expand table

Name	Description
collection_name Required*	str The collection name.
records Required*	<xref:List>[<xref:MemoryRecord>] A list of memory records.
batch_size	<xref:<xref:semantic_kernel.connectors.memory.milvus.milvus_memory_store.int, optional>> Batch size of the insert, 0 is a batch size of total size. Defaults to 100. default value: 100

Returns

[] Expand table

Type	Description
List[str]	A list of inserted ID's.

Exceptions

[] Expand table

Type	Description
Exception	Collection doesn't exist.

Type	Description
e	Failed to upsert a record.

Attributes

collections

Python

```
collections: dict[str, pymilvus.orm.collection.Collection]
```

is_experimental

Python

```
is_experimental = True
```

MilvusMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

MilvusMemoryStore allows for searching for records using Milvus/Zilliz Cloud.

For more details on how to get the service started, take a look here: Milvus:

https://milvus.io/docs/get_started.md ↗ Zilliz Cloud: <https://docs.zilliz.com/docs/quick-start> ↗

Inheritance [MemoryStoreBase](#) → MilvusMemoryStore

Constructor

Python

```
MilvusMemoryStore(uri: str = 'http://localhost:19530', token: str | None = None, **kwargs)
```

Parameters

[+] Expand table

Name	Description
uri	<xref:<xref:semantic_kernel.connectors.memory.milvus.str, optional>> The uri of the cluster. Defaults to "http://localhost:19530". default value: http://localhost:19530
token	<xref:Optional>[str],<xref: optional> The token to connect to the cluster if authentication is required. Defaults to None. default value: None

Methods

[+] Expand table

create_collection	Create a Milvus collection.
delete_collection	Delete the specified collection.

	If all is True, all collections in the cluster will be removed.
does_collection_exist	Return if the collection exists in the cluster.
get	Get the MemoryRecord corresponding to the key.
get_batch	Get the MemoryRecords corresponding to the keys
get_collections	Return a list of present collections.
get_nearest_match	Find the nearest match for an embedding.
get_nearest_matches	Find the nearest <i>limit</i> matches for an embedding.
remove	Remove the specified record based on key.
remove_batch	Remove multiple records based on keys.
upsert	Upsert a single MemoryRecord into the collection.
upsert_batch	<i>summary</i>

create_collection

Create a Milvus collection.

Python

```
async create_collection(collection_name: str, dimension_num: int = 1536,
distance_type: str | None = 'IP', overwrite: bool = False, consistency:
str = 'Session') -> None
```

Parameters

[+] [Expand table](#)

Name	Description
collection_name Required*	str The name of the collection.
dimension_num	<xref:Optional>[int],<xref: optional> The size of the embeddings being stored. Defaults to 1536. default value: 1536
distance_type	<xref:Optional>[str],<xref: optional> Which distance function, at the moment only "IP" and "L2" are supported. Defaults to "IP".

Name	Description
	default value: IP
overwrite	<xref:<xref:semantic_kernel.connectors.memory.milvus.bool, optional>> Whether to overwrite any existing collection with the same name. Defaults to False. default value: False
consistency	<xref:<xref:semantic_kernel.connectors.memory.milvus.str, optional>> Which consistency level to use: Strong, Session, Bounded, Eventually. Defaults to "Session". default value: Session

delete_collection

Delete the specified collection.

If all is True, all collections in the cluster will be removed.

Python

```
async delete_collection(collection_name: str | None = None, all: bool = False) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name	<xref:<xref:semantic_kernel.connectors.memory.milvus.str, optional>> The name of the collection to delete. Defaults to "". default value: None
all	<xref:<xref:semantic_kernel.connectors.memory.milvus.bool, optional>> Whether to delete all collections. Defaults to False. default value: False

does_collection_exist

Return if the collection exists in the cluster.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str The name of the collection.

Returns

[+] Expand table

Type	Description
bool	True if it exists, False otherwise.

get

Get the MemoryRecord corresponding to the key.

Python

```
async get(collection_name: str, key: str, with_embedding: bool) ->
MemoryRecord
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str The collection to get from.
key Required*	str The ID to grab.
with_embedding Required*	bool Whether to include the embedding in the results.

Returns

[+] Expand table

Type	Description
MemoryRecord	The MemoryRecord for the key.

get_batch

Get the MemoryRecords corresponding to the keys

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str <i>description</i>
keys Required*	<xref>List>[str] <i>description</i>
with_embeddings Required*	bool <i>description</i>

Returns

[+] Expand table

Type	Description
List[MemoryRecord]	<i>description</i>

Exceptions

[\[\] Expand table](#)

Type	Description
Exception	<i>description</i>
e	<i>description</i>

get_collections

Return a list of present collections.

Python

```
async get_collections() -> list[str]
```

Returns

[\[\] Expand table](#)

Type	Description
List[str]	List of collection names.

get_nearest_match

Find the nearest match for an embedding.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float = 0.0, with_embedding: bool = False) ->  
tuple[semantic_kernel.memory.MemoryRecord, float]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name	<code>str</code> Required*

Name	Description
embedding Required*	<xref:semantic_kernel.connectors.memory.milvus.ndarray> The embedding to search for.
min_relevance_score	<xref:<xref:semantic_kernel.connectors.memory.milvus.float, optional>> 20. Defaults to 0.0. default value: 0.0
with_embedding	<xref:<xref:semantic_kernel.connectors.memory.milvus.bool, optional>> Whether to include embedding in result. Defaults to False. default value: False

Returns

[+] Expand table

Type	Description
<code>Tuple[MemoryRecord, float]</code>	A tuple of record and distance.

get_nearest_matches

Find the nearest *limit* matches for an embedding.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,
limit: int, min_relevance_score: float = 0.0, with_embeddings: bool =
False) -> list[tuple[semantic_kernel.memory.memory_record.MemoryRecord,
float]]
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	<code>str</code> The collection to search.

Name	Description
embedding Required*	<xref:semantic_kernel.connectors.memory.milvus.ndarray> The embedding to search.
limit Required*	int The total results to display.
min_relevance_score	<xref:<xref:semantic_kernel.connectors.memory.milvus.float, optional>> Minimum distance to include. Defaults to None. default value: 0.0
with_embeddings	<xref:<xref:semantic_kernel.connectors.memory.milvus.bool, optional>> Whether to include embeddings in result. Defaults to False. default value: False

Returns

[+] Expand table

Type	Description
List[Tuple[MemoryRecord, float]]	MemoryRecord and distance tuple.

Exceptions

[+] Expand table

Type	Description
Exception	Missing collection
e	Failure to search

remove

Remove the specified record based on key.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str Collection to remove from.
key Required*	str The key to remove.

remove_batch

Remove multiple records based on keys.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str Collection to remove from
keys Required*	<xref>List>[str] The list of keys.

Exceptions

[+] Expand table

Type	Description
Exception	Collection doesn't exist.
e	Failure to remove key.

upsert

Upsert a single MemoryRecord into the collection.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str The name of the collection.
record Required*	<xref:semantic_kernel.connectors.memory.milvus.MemoryRecord> The record to store.

Returns

[+] Expand table

Type	Description
str	The ID of the inserted record.

upsert_batch

summary

Python

```
async upsert_batch(collection_name: str, records:
list[semantic_kernel.memory.memory_record.MemoryRecord], batch_size=100)
-> list[str]
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	<code>str</code> The collection name.
records Required*	<code><xref>List>[<xref:MemoryRecord>]</code> A list of memory records.
batch_size	<code><xref:<xref:semantic_kernel.connectors.memory.milvus.int, optional>></code> Batch size of the insert, 0 is a batch size of total size. Defaults to 100. default value: 100

Returns

[\[\] Expand table](#)

Type	Description
<code>List[str]</code>	A list of inserted ID's.

Exceptions

[\[\] Expand table](#)

Type	Description
<code>Exception</code>	Collection doesnt exist.
<code>e</code>	Failed to upsert a record.

Attributes

is_experimental

Python

```
is_experimental = True
```

qdrant Package

Reference

Modules

[+] [Expand table](#)

qdrant_memory_store	QdrantMemoryStore provides functionality to add Qdrant vector database to support Semantic Kernel memory. The QdrantMemoryStore inherits from MemoryStoreBase for persisting/retrieving data from a Qdrant Vector Database.
-------------------------------------	---

Classes

[+] [Expand table](#)

QdrantMemoryStore	Note: This class is experimental and may change in the future. Initializes a new instance of the QdrantMemoryStore class.
-----------------------------------	--

qdrant_memory_store Module

Reference

QdrantMemoryStore provides functionality to add Qdrant vector database to support Semantic Kernel memory. The QdrantMemoryStore inherits from MemoryStoreBase for persisting/retrieving data from a Qdrant Vector Database.

Classes

[] [Expand table](#)

[QdrantMemoryStore](#)

Note: This class is experimental and may change in the future.

Initializes a new instance of the QdrantMemoryStore class.

QdrantMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

Initializes a new instance of the QdrantMemoryStore class.

Inheritance [MemoryStoreBase](#) → QdrantMemoryStore

Constructor

Python

```
QdrantMemoryStore(vector_size: int, url: str | None = None, port: int | None = 6333, local: bool | None = False, **kwargs)
```

Parameters

[\[+\] Expand table](#)

Name	Description
vector_size Required*	
url	default value: None
port	default value: 6333
local	default value: False

Methods

[\[+\] Expand table](#)

create_collection	Creates a new collection if it does not exist.
delete_collection	Deletes a collection.
does_collection_exist	Checks if a collection exists.
get	

get_batch	
get_collection	Gets the a collections based upon collection name.
get_collections	Gets the list of collections.
get_nearest_match	
get_nearest_matches	
remove	
remove_batch	
upsert	Upserts a record.
upsert_batch	

create_collection

Creates a new collection if it does not exist.

Python

```
async create_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
create. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
vector. Required*	<xref:vector_size {int} -- The size> of <xref:the>
collection_name Required*	

Returns

[+] Expand table

Type	Description
	None

delete_collection

Deletes a collection.

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
delete. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

does_collection_exist

Checks if a collection exists.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[\[\] Expand table](#)

Name	Description
check. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	bool – True if the collection exists; otherwise, False.

get

Python

```
async get(collection_name: str, key: str, with_embedding: bool = False) -> MemoryRecord | None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	
with_embedding	default value: False

get_batch

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool = False) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	
keys Required*	
with_embeddings	default value: False

get_collection

Gets the a collections based upon collection name.

Python

```
async get_collection(collection_name: str) -> CollectionInfo
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

Returns

[+] Expand table

Type	Description
	CollectionInfo – Collection Information from Qdrant about collection.

get_collections

Gets the list of collections.

Python

```
async get_collections() -> list[str]
```

Returns

[\[\] Expand table](#)

Type	Description
	List[str] – The list of collections.

get_nearest_match

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float, with_embedding: bool = False) ->  
tuple[semantic_kernel.memory.MemoryRecord, float]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
embedding Required*	
min_relevance_score Required*	
with_embedding	default value: False

get_nearest_matches

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,  
limit: int, min_relevance_score: float, with_embeddings: bool = False) ->  
list[tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score Required*	
with_embeddings	default value: False

remove

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	

remove_batch

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	
keys Required*	

upsert

Upserts a record.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[+] Expand table

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the record>>
upsert. Required*	<xref:<xref:record {MemoryRecord} -- The record to>>
collection_name Required*	
record Required*	

Returns

[\[\] Expand table](#)

Type	Description
	str – The unique database key of the record.

upsert_batch

Python

```
async upsert_batch(collection_name: str, records:  
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
records Required*	

Attributes

is_experimental

Python

```
is_experimental = True
```

QdrantMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

Initializes a new instance of the QdrantMemoryStore class.

Inheritance [MemoryStoreBase](#) → QdrantMemoryStore

Constructor

Python

```
QdrantMemoryStore(vector_size: int, url: str | None = None, port: int | None = 6333, local: bool | None = False, **kwargs)
```

Parameters

[\[+\] Expand table](#)

Name	Description
vector_size Required*	
url	default value: None
port	default value: 6333
local	default value: False

Methods

[\[+\] Expand table](#)

create_collection	Creates a new collection if it does not exist.
delete_collection	Deletes a collection.
does_collection_exist	Checks if a collection exists.
get	

get_batch	
get_collection	Gets the a collections based upon collection name.
get_collections	Gets the list of collections.
get_nearest_match	
get_nearest_matches	
remove	
remove_batch	
upsert	Upserts a record.
upsert_batch	

create_collection

Creates a new collection if it does not exist.

Python

```
async create_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
create. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
vector. Required*	<xref:vector_size {int} -- The size> of <xref:the>
collection_name Required*	

Returns

[+] Expand table

Type	Description
	None

delete_collection

Deletes a collection.

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
delete. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

does_collection_exist

Checks if a collection exists.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[\[\] Expand table](#)

Name	Description
check. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	bool – True if the collection exists; otherwise, False.

get

Python

```
async get(collection_name: str, key: str, with_embedding: bool = False) -> MemoryRecord | None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	
with_embedding	default value: False

get_batch

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool = False) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	
keys Required*	
with_embeddings	default value: False

get_collection

Gets the a collections based upon collection name.

Python

```
async get_collection(collection_name: str) -> CollectionInfo
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

Returns

[+] Expand table

Type	Description
	CollectionInfo – Collection Information from Qdrant about collection.

get_collections

Gets the list of collections.

Python

```
async get_collections() -> list[str]
```

Returns

[\[\] Expand table](#)

Type	Description
	List[str] – The list of collections.

get_nearest_match

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
min_relevance_score: float, with_embedding: bool = False) ->  
tuple[semantic_kernel.memory.MemoryRecord, float]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
embedding Required*	
min_relevance_score Required*	
with_embedding	default value: False

get_nearest_matches

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,  
limit: int, min_relevance_score: float, with_embeddings: bool = False) ->  
list[tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score Required*	
with_embeddings	default value: False

remove

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	

remove_batch

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	
keys Required*	

upsert

Upserts a record.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[+] Expand table

Name	Description
into. Required*	<xref:<xref:collection_name {str} -- The name> of <xref:the collection to upsert the record>>
upsert. Required*	<xref:<xref:record {MemoryRecord} -- The record to>>
collection_name Required*	
record Required*	

Returns

[\[\] Expand table](#)

Type	Description
	str – The unique database key of the record.

upsert_batch

Python

```
async upsert_batch(collection_name: str, records:  
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
records Required*	

Attributes

is_experimental

Python

```
is_experimental = True
```

usearch Package

Reference

Modules

[] [Expand table](#)

[usearch_memory_store](#)

Classes

[] [Expand table](#)

[USearchMemoryStore](#) Note: This class is experimental and may change in the future.

Create a USearchMemoryStore instance.

This store helps searching embeddings with USearch, keeping collections in memory. To save collections to disk, provide the *persist_directory* param. Collections are saved when *close* is called.

To both save collections and free up memory, call *close*. When *USearchMemoryStore* is used with a context manager, this will happen automatically. Otherwise, it should be called explicitly.

usearch_memory_store Module

Reference

Classes

[+] Expand table

[USearchMemoryStore](#) Note: This class is experimental and may change in the future.

Create a USearchMemoryStore instance.

This store helps searching embeddings with USearch, keeping collections in memory. To save collections to disk, provide the *persist_directory* param. Collections are saved when *close* is called.

To both save collections and free up memory, call *close*. When *USearchMemoryStore* is used with a context manager, this will happen automatically. Otherwise, it should be called explicitly.

Functions

memoryrecords_to_pyarrow_table

Convert a list of *MemoryRecord* to a PyArrow Table

Python

```
memoryrecords_to_pyarrow_table(records:  
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> Table
```

Parameters

[+] Expand table

Name	Description
records Required*	

pyarrow_table_to_memoryrecords

Convert a PyArrow Table to a list of MemoryRecords.

Python

```
pyarrow_table_to_memoryrecords(table: Table, vectors: ndarray | None = None) -> list[semantic_kernel.memory.MemoryRecord]
```

Parameters

[Expand table

Name	Description
table Required*	<xref:pa.Table> The PyArrow Table to convert.
vectors	<xref:Optional>[<xref:ndarray>],<xref: optional> An array of vectors to include as embeddings in the MemoryRecords. The length and order of the vectors should match the rows in the table. Defaults to None. default value: None

Returns

[Expand table

Type	Description
List[MemoryRecord]	List of MemoryRecords constructed from the table.

USearchMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

Create a USearchMemoryStore instance.

This store helps searching embeddings with USearch, keeping collections in memory. To save collections to disk, provide the *persist_directory* param. Collections are saved when *close* is called.

To both save collections and free up memory, call *close*. When *USearchMemoryStore* is used with a context manager, this will happen automatically. Otherwise, it should be called explicitly.

Inheritance [MemoryStoreBase](#) → USearchMemoryStore

Constructor

Python

```
USearchMemoryStore(persist_directory: PathLike | None = None)
```

Parameters

[Expand table](#)

Name	Description
persist_directory	<xref:Optional>[PathLike],<xref: default=None> Directory for loading and saving collections. default value: None
None Required*	<xref:If>
saved. Required*	<xref:<xref:collections are not loaded nor>>

Methods

[Expand table](#)

close	Persist collection, clear.
-----------------------	----------------------------

create_collection	Create a new collection.
delete_collection	
does_collection_exist	
get	Retrieve a single MemoryRecord using its key.
get_batch	Retrieve a batch of MemoryRecords using their keys.
get_collections	Get list of existing collections.
get_nearest_match	<p>Retrieve the nearest matching MemoryRecord for the provided embedding.</p> <p>By default it is approximately search, see <i>exact</i> param description.</p> <p>Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.</p>
get_nearest_matches	<p>Get the nearest matches to a given embedding.</p> <p>By default it is approximately search, see <i>exact</i> param description.</p> <p>Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.</p>
remove	Remove a single MemoryRecord using its key.
remove_batch	Remove a batch of MemoryRecords using their keys.
upsert	Upsert single MemoryRecord and return its ID.
upsert_batch	Upsert a batch of MemoryRecords and return their IDs.

close

Persist collection, clear.

Python

```
async close() -> None
```

Returns

[\[\]](#) Expand table

Type	Description
	None

create_collection

Create a new collection.

Python

```
async create_collection(collection_name: str, ndim: int = 0, metric: str | ~usearch.compiled.MetricKind | ~usearch.index.CompiledMetric = <MetricKind.IP: 105>, dtype: str | ~usearch.compiled.ScalarKind | None = None, connectivity: int | None = None, expansion_add: int | None = None, expansion_search: int | None = None, view: bool = False) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str Name of the collection. Case-insensitive. Must have name that is valid file name for the current OS environment.
ndim	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Number of dimensions. Defaults to 0. default value: 0
metric	<xref:Union>[str,<xref: MetricKind>,<xref: CompiledMetric>],<xref: optional> Metric kind. Defaults to MetricKind.IP. default value: MetricKind.IP
dtype	<xref:Optional>[<xref:Union>[str,<xref: ScalarKind>]],<xref: optional> Data type. Defaults to None. default value: None
connectivity	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Connectivity parameter. Defaults to None. default value: None
expansion_add	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Expansion add parameter. Defaults to None. default value: None
expansion_search	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Expansion search parameter. Defaults to None.

Name	Description
	default value: None
view	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> Viewing flag. Defaults to False. default value: False

Exceptions

[Expand table](#)

Type	Description
ValueError	If collection with the given name already exists.
ValueError	If collection name is empty string.

delete_collection

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	

does_collection_exist

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	

get

Retrieve a single MemoryRecord using its key.

Python

```
async get(collection_name: str, key: str, with_embedding: bool, dtype: ~usearch.compiled.ScalarKind = <ScalarKind.F32: 11>) -> MemoryRecord
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	
key Required*	
with_embedding Required*	
dtype	default value: ScalarKind.F32

get_batch

Retrieve a batch of MemoryRecords using their keys.

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool, dtype: ~usearch.compiled.ScalarKind = <ScalarKind.F32: 11>) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	
keys Required*	
with_embeddings Required*	
dtype	default value: ScalarKind.F32

get_collections

Get list of existing collections.

Python

```
async get_collections() -> list[str]
```

Returns

[+] [Expand table](#)

Type	Description
List[str]	List of collection names.

get_nearest_match

Retrieve the nearest matching MemoryRecord for the provided embedding.

By default it is approximately search, see *exact* param description.

Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,
min_relevance_score: float = 0.0, with_embedding: bool = True, exact: bool =
False) -> tuple[semantic_kernel.memory.MemoryRecord, float]
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str Name of the collection to search within.
embedding Required*	<xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.ndarray> The embedding vector to search for.
min_relevance_score	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.float, optional>> The minimum relevance score for vectors. Supposed to be from 0 to 1. Only vectors with greater or equal relevance score are returned. Defaults to 0.0. default value: 0.0
with_embedding	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> If True, include the embedding in the result. Defaults to True. default value: True
exact	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> Perform exhaustive linear-time exact search. Defaults to False. default value: False

Returns

[Expand table](#)

Type	Description
Tuple[MemoryRecord, float]	The nearest matching record and its relevance score.

get_nearest_matches

Get the nearest matches to a given embedding.

By default it is approximately search, see *exact* param description.

Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray, limit: int,
                           min_relevance_score: float = 0.0, with_embeddings: bool = True, *, threads: int
```

```
= 0, exact: bool = False, log: str | bool = False, batch_size: int = 0) ->
list[tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]]
```

Parameters

[\[+\] Expand table](#)

Name	Description
collection_name Required*	str Name of the collection to search within.
embedding Required*	<xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.ndarray> The embedding vector to search for.
limit Required*	int maximum amount of embeddings to search for.
min_relevance_score	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.float, optional>> The minimum relevance score for vectors. Supposed to be from 0 to 1. Only vectors with greater or equal relevance score are returned. Defaults to 0.0. default value: 0.0
with_embedding Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> If True, include the embedding in the result. Defaults to True.
threads Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Optimal number of cores to use. Defaults to 0.
exact Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> Perform exhaustive linear-time exact search. Defaults to False.
log Required*	<xref:Union>[str, bool], <xref: optional> Whether to print the progress bar. Defaults to False.
batch_size Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Number of vectors to process at once. Defaults to 0.
with_embeddings	default value: True

Keyword-Only Parameters

[Expand table](#)

Name	Description
threads Required*	
exact Required*	
log Required*	
batch_size Required*	

Returns

[Expand table](#)

Type	Description
<code>List[Tuple[MemoryRecord, float]]</code>	The nearest matching records and their relevance score.

Exceptions

[Expand table](#)

Type	Description
<code>KeyError</code>	if a collection with specified name does not exist

remove

Remove a single MemoryRecord using its key.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	
key Required*	

remove_batch

Remove a batch of MemoryRecords using their keys.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	
keys Required*	

upsert

Upsert single MemoryRecord and return its ID.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	
record	

Name	Description
Required*	

upsert_batch

Upsert a batch of MemoryRecords and return their IDs.

Python

```
async upsert_batch(collection_name: str, records:
list[semantic_kernel.memory.memory_record.MemoryRecord], *, compact: bool =
False, copy: bool = True, threads: int = 0, log: str | bool = False,
batch_size: int = 0) -> list[str]
```

Parameters

[] Expand table

Name	Description
collection_name Required*	str Name of the collection to search within.
records Required*	<xref:List>[<xref:MemoryRecord>] Records to upsert.
compact Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> Removes links to removed nodes (expensive). Defaults to False.
copy Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.bool, optional>> Should the index store a copy of vectors. Defaults to True.
threads Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Optimal number of cores to use. Defaults to 0.
log Required*	<xref:Union>[str,bool],<xref: optional> Whether to print the progress bar. Defaults to False.
batch_size Required*	<xref: <xref:semantic_kernel.connectors.memory.usearch.usearch_memory_store.int, optional>> Number of vectors to process at once. Defaults to 0.

Keyword-Only Parameters

[Expand table](#)

Name	Description
compact Required*	
copy	default value: True
threads Required*	
log Required*	
batch_size Required*	

Returns

[Expand table](#)

Type	Description
List[str]	List of IDs.

Exceptions

[Expand table](#)

Type	Description
KeyError	If collection not exist

Attributes

is_experimental

Python

```
is_experimental = True
```

USearchMemoryStore Class

Reference

Note: This class is experimental and may change in the future.

Create a USearchMemoryStore instance.

This store helps searching embeddings with USearch, keeping collections in memory. To save collections to disk, provide the `persist_directory` param. Collections are saved when `close` is called.

To both save collections and free up memory, call `close`. When `USearchMemoryStore` is used with a context manager, this will happen automatically. Otherwise, it should be called explicitly.

Inheritance [MemoryStoreBase](#) → USearchMemoryStore

Constructor

Python

```
USearchMemoryStore(persist_directory: PathLike | None = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
<code>persist_directory</code>	<xref:Optional> [PathLike], <xref: default=None> Directory for loading and saving collections. default value: None
<code>None</code> Required*	<xref:If>
<code>saved.</code> Required*	<xref:><xref: collections are not loaded nor>>

Methods

close	Persist collection, clear.
create_collection	Create a new collection.
delete_collection	
does_collection_exist	
get	Retrieve a single MemoryRecord using its key.
get_batch	Retrieve a batch of MemoryRecords using their keys.
get_collections	Get list of existing collections.
get_nearest_match	<p>Retrieve the nearest matching MemoryRecord for the provided embedding.</p> <p>By default it is approximately search, see <i>exact</i> param description.</p> <p>Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.</p>
get_nearest_matches	<p>Get the nearest matches to a given embedding.</p> <p>By default it is approximately search, see <i>exact</i> param description.</p> <p>Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.</p>
remove	Remove a single MemoryRecord using its key.
remove_batch	Remove a batch of MemoryRecords using their keys.
upsert	Upsert single MemoryRecord and return its ID.
upsert_batch	Upsert a batch of MemoryRecords and return their IDs.

close

Persist collection, clear.

Python

```
async close() -> None
```

Returns

[+] Expand table

Type	Description
	None

create_collection

Create a new collection.

Python

```
async create_collection(collection_name: str, ndim: int = 0, metric: str | ~usearch.compiled.MetricKind | ~usearch.index.CompiledMetric = <MetricKind.IP: 105>, dtype: str | ~usearch.compiled.ScalarKind | None = None, connectivity: int | None = None, expansion_add: int | None = None, expansion_search: int | None = None, view: bool = False) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	str Name of the collection. Case-insensitive. Must have name that is valid file name for the current OS environment.
ndim	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Number of dimensions. Defaults to 0. default value: 0
metric	<xref:Union>[str ,<xref: MetricKind>,<xref: CompiledMetric>],<xref: optional> Metric kind. Defaults to MetricKind.IP. default value: MetricKind.IP
dtype	<xref:Optional>[<xref:Union>[str ,<xref: ScalarKind>]],<xref: optional> Data type. Defaults to None. default value: None
connectivity	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Connectivity parameter. Defaults to None.

Name	Description
	default value: None
expansion_add	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Expansion add parameter. Defaults to None. default value: None
expansion_search	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Expansion search parameter. Defaults to None. default value: None
view	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> Viewing flag. Defaults to False. default value: False

Exceptions

[+] Expand table

Type	Description
ValueError	If collection with the given name already exists.
ValueError	If collection name is empty string.

delete_collection

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

does_collection_exist

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[\[+\] Expand table](#)

Name	Description
collection_name Required*	

get

Retrieve a single MemoryRecord using its key.

Python

```
async get(collection_name: str, key: str, with_embedding: bool, dtype: ~usearch.compiled.ScalarKind = <ScalarKind.F32: 11>) -> MemoryRecord
```

Parameters

[\[+\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	
with_embedding Required*	
dtype	default value: ScalarKind.F32

get_batch

Retrieve a batch of MemoryRecords using their keys.

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool, dtype: ~usearch.compiled.ScalarKind = <ScalarKind.F32: 11>) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[Expand table

Name	Description
collection_name Required*	
keys Required*	
with_embeddings Required*	
dtype	default value: ScalarKind.F32

get_collections

Get list of existing collections.

Python

```
async get_collections() -> list[str]
```

Returns

[Expand table

Type	Description
List[str]	List of collection names.

get_nearest_match

Retrieve the nearest matching MemoryRecord for the provided embedding.

By default it is approximately search, see *exact* param description.

Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,
min_relevance_score: float = 0.0, with_embedding: bool = True, exact:
bool = False) -> tuple[semantic_kernel.memory.memory_record.MemoryRecord,
float]
```

Parameters

[Expand table](#)

Name	Description
collection_name Required*	str Name of the collection to search within.
embedding Required*	<xref:semantic_kernel.connectors.memory.usearch.ndarray> The embedding vector to search for.
min_relevance_score	<xref:<xref:semantic_kernel.connectors.memory.usearch.float, optional>> The minimum relevance score for vectors. Supposed to be from 0 to 1. Only vectors with greater or equal relevance score are returned. Defaults to 0.0. default value: 0.0
with_embedding	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> If True, include the embedding in the result. Defaults to True. default value: True
exact	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> Perform exhaustive linear-time exact search. Defaults to False. default value: False

Returns

[\[\] Expand table](#)

Type	Description
<code>Tuple[MemoryRecord, float]</code>	The nearest matching record and its relevance score.

get_nearest_matches

Get the nearest matches to a given embedding.

By default it is approximately search, see *exact* param description.

Measure of similarity between vectors is relevance score. It is from 0 to 1. USearch returns distances for vectors. Distance is converted to relevance score by inverse function.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,
limit: int, min_relevance_score: float = 0.0, with_embeddings: bool =
True, *, threads: int = 0, exact: bool = False, log: str | bool = False,
batch_size: int = 0) ->
list[tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	<code>str</code> Name of the collection to search within.
embedding Required*	<code><xref:semantic_kernel.connectors.memory.usearch.ndarray></code> The embedding vector to search for.
limit Required*	<code>int</code> maximum amount of embeddings to search for.
min_relevance_score	<code><xref:<xref:semantic_kernel.connectors.memory.usearch.float, optional>></code> The minimum relevance score for vectors. Supposed to be from 0 to 1. Only vectors with greater or equal relevance score are returned. Defaults to 0.0. default value: 0.0

Name	Description
with_embedding Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> If True, include the embedding in the result. Defaults to True.
threads Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Optimal number of cores to use. Defaults to 0.
exact Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> Perform exhaustive linear-time exact search. Defaults to False.
log Required*	<xref:Union>[str,bool],<xref: optional> Whether to print the progress bar. Defaults to False.
batch_size Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Number of vectors to process at once. Defaults to 0.
with_embeddings	default value: True

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
threads Required*	
exact Required*	
log Required*	
batch_size Required*	

Returns

[\[\] Expand table](#)

Type	Description
<code>List[Tuple[MemoryRecord, float]]</code>	The nearest matching records and their relevance score.

Exceptions

[\[\] Expand table](#)

Type	Description
<code>KeyError</code>	if a collection with specified name does not exist

remove

Remove a single MemoryRecord using its key.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>collection_name</code> Required*	
<code>key</code> Required*	

remove_batch

Remove a batch of MemoryRecords using their keys.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
keys Required*	

upsert

Upsert single MemoryRecord and return its ID.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
record Required*	

upsert_batch

Upsert a batch of MemoryRecords and return their IDs.

Python

```
async upsert_batch(collection_name: str, records: list[semantic_kernel.memory.memory_record.MemoryRecord], *, compact: bool = False, copy: bool = True, threads: int = 0, log: str | bool = False, batch_size: int = 0) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	str Name of the collection to search within.
records Required*	<xref>List>[<xref:MemoryRecord>] Records to upsert.
compact Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> Removes links to removed nodes (expensive). Defaults to False.
copy Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.bool, optional>> Should the index store a copy of vectors. Defaults to True.
threads Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Optimal number of cores to use. Defaults to 0.
log Required*	<xref:Union>[str, bool],<xref: optional> Whether to print the progress bar. Defaults to False.
batch_size Required*	<xref:<xref:semantic_kernel.connectors.memory.usearch.int, optional>> Number of vectors to process at once. Defaults to 0.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
compact Required*	
copy	default value: True
threads Required*	
log Required*	
batch_size Required*	

Returns

[\[\]](#) Expand table

Type	Description
List[str]	List of IDs.

Exceptions

[\[\]](#) Expand table

Type	Description
KeyError	If collection not exist

Attributes

is_experimental

Python

```
is_experimental = True
```

utils Module

Reference

Classes

[+] Expand table

[AsyncSession](#)

Functions

build_payload

Builds a metadata payload to be sent to AstraDb from a MemoryRecord.

Python

```
build_payload(record: MemoryRecord) -> dict[str, Any]
```

Parameters

[+] Expand table

Name	Description
record Required*	

parse_payload

Parses a record from AstraDb into a MemoryRecord.

Python

```
parse_payload(document: dict[str, Any]) -> MemoryRecord
```

Parameters

[] Expand table

Name	Description
document Required*	

AsyncSession Class

Reference

Inheritance builtins.object → AsyncSession

Constructor

Python

```
AsyncSession(session: ClientSession = None)
```

Parameters

[+] [Expand table](#)

Name	Description
session	default value: None

azure_cosmos_db_store_api Module

Reference

Classes

 Expand table

[AzureCosmosDBStoreApi](#)

Note: This class is experimental and may change in the future.

AzureCosmosDBStoreApi Class

Reference

Note: This class is experimental and may change in the future.

Inheritance [ABC](#) → AzureCosmosDBStoreApi

Constructor

Python

```
AzureCosmosDBStoreApi()
```

Methods

[\[\]](#) [Expand table](#)

[create_collection](#)

[delete_collection](#)

[does_collection_exist](#)

[get](#)

[get_batch](#)

[get_collections](#)

[get_nearest_match](#)

[get_nearest_matches](#)

[remove](#)

[remove_batch](#)

[upsert](#)

[upsert_batch](#)

create_collection

Python

```
abstract async create_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

delete_collection

Python

```
abstract async delete_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

does_collection_exist

Python

```
abstract async does_collection_exist(collection_name: str) -> bool
```

Parameters

[+] Expand table

Name	Description
collection_name Required*	

get

Python

```
abstract async get(collection_name: str, key: str, with_embedding: bool)  
-> MemoryRecord
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	
with_embedding Required*	

get_batch

Python

```
abstract async get_batch(collection_name: str, keys: list[str],  
with_embeddings: bool) ->  
list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name	

Name	Description
Required*	
keys	
Required*	
with_embeddings	
Required*	

get_collections

Python

```
abstract async get_collections() -> list[str]
```

get_nearest_match

Python

```
abstract async get_nearest_match(collection_name: str, embedding: ndarray, min_relevance_score: float, with_embedding: bool) -> tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name	
Required*	
embedding	
Required*	
min_relevance_score	
Required*	
with_embedding	
Required*	

get_nearest_matches

Python

```
abstract async get_nearest_matches(collection_name: str, embedding: ndarray, limit: int, min_relevance_score: float, with_embeddings: bool) -> list[tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score Required*	
with_embeddings Required*	

remove

Python

```
abstract async remove(collection_name: str, key: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
key Required*	

remove_batch

Python

```
abstract async remove_batch(collection_name: str, keys: list[str]) -> None
```

[\[\] Expand table](#)

Name	Description
collection_name Required*	
keys Required*	

upsert

Python

```
abstract async upsert(collection_name: str, record: MemoryRecord) -> str
```

[\[\] Expand table](#)

Name	Description
collection_name Required*	
record Required*	

upsert_batch

Python

```
abstract async upsert_batch(collection_name: str, records:  
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
collection_name Required*	
records Required*	

Attributes

is_experimental

Python

```
is_experimental = True
```

memory_settings_base Module

Reference

Classes

[] [Expand table](#)

[BaseModelSettings](#)

Note: This class is experimental and may change in the future.

BaseModelSettings Class

Reference

Note: This class is experimental and may change in the future.

Inheritance `pydantic_settings.main.BaseSettings` → `BaseModelSettings`

Constructor

Python

```
BaseModelSettings(_case_sensitive: bool | None = None, _env_prefix: str | None = None, _env_file: DotenvType | None = WindowsPath('.'), _env_file_encoding: str | None = None, _env_ignore_empty: bool | None = None, _env_nested_delimiter: str | None = None, _env_parse_none_str: str | None = None, _secrets_dir: str | Path | None = None, *, env_file_path: str | None = None)
```

Parameters

[+] Expand table

Name	Description
<code>_case_sensitive</code>	default value: None
<code>_env_prefix</code>	default value: None
<code>_env_file</code>	default value: .
<code>_env_file_encoding</code>	default value: None
<code>_env_ignore_empty</code>	default value: None
<code>_env_nested_delimiter</code>	default value: None
<code>_env_parse_none_str</code>	default value: None
<code>_secrets_dir</code>	default value: None

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>env_file_path</code> Required*	

Methods

[] Expand table

[create](#)

create

Python

```
create(**kwargs)
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[SettingsConfigDict] = { 'arbitrary_types_allowed': True, 'case_sensitive': False, 'env_file': None, 'env_file_encoding': 'utf-8', 'env_ignore_empty': False, 'env_nested_delimiter': None, 'env_parse_none_str': None, 'env_prefix': '', 'extra': 'ignore', 'json_file': None, 'json_file_encoding': None, 'protected_namespaces': [] }
```

```
('model_', 'settings_'), 'secrets_dir': None, 'toml_file': None,  
'validate_default': True, 'yaml_file': None, 'yaml_file_encoding': None}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'env_file_path':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

env_file_path

Python

```
env_file_path: str | None
```

is_experimental

Python

```
is_experimental = True
```

Config Class

Reference

Inheritance builtins.object → Config

Constructor

Python

```
Config()
```

Attributes

case_sensitive

Python

```
case_sensitive = False
```

env_file

Python

```
env_file = None
```

env_file_encoding

Python

```
env_file_encoding = 'utf-8'
```

extra

Python

```
extra = 'ignore'
```


mongodb_atlas_memory_store Module

Reference

Classes

 Expand table

[MongoDBAtlasMemoryStore](#)

Memory Store for MongoDB Atlas Vector Search Connections

Note: This class is experimental and may change in the future.

MongoDBAtlasMemoryStore Class

Reference

Memory Store for MongoDB Atlas Vector Search Connections

Note: This class is experimental and may change in the future.

Inheritance [MemoryStoreBase](#) → MongoDBAtlasMemoryStore

Constructor

Python

```
MongoDBAtlasMemoryStore(index_name: str | None = None, connection_string: str | None = None, database_name: str | None = None, read_preference: ReadPreference | None = Primary(), env_file_path: str | None = None)
```

Parameters

[\[+\] Expand table](#)

Name	Description
<code>index_name</code>	default value: None
<code>connection_string</code>	default value: None
<code>database_name</code>	default value: None
<code>read_preference</code>	default value: Primary()
<code>env_file_path</code>	default value: None

Methods

[\[+\] Expand table](#)

<code>close</code>	Async close connection, invoked by <code>MemoryStoreBase.aexit()</code>
<code>create_collection</code>	Creates a new collection in the data store.
<code>delete_collection</code>	Deletes a collection from the data store.

<code>does_collection_exist</code>	Determines if a collection exists in the data store.
<code>get</code>	Gets a memory record from the data store. Does not guarantee that the collection exists.
<code>get_batch</code>	Gets a batch of memory records from the data store. Does not guarantee that the collection exists.
<code>get_collections</code>	Gets all collection names in the data store.
<code>get_nearest_match</code>	Gets the nearest match to an embedding of type float. Does not guarantee that the collection exists.
<code>get_nearest_matches</code>	Gets the nearest matches to an embedding of type float. Does not guarantee that the collection exists.
<code>remove</code>	Removes a memory record from the data store. Does not guarantee that the collection exists.
<code>remove_batch</code>	Removes a batch of memory records from the data store. Does not guarantee that the collection exists.
<code>upsert</code>	Upserts a memory record into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.
<code>upsert_batch</code>	Upserts a group of memory records into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

close

Async close connection, invoked by `MemoryStoreBase.aexit()`

Python

```
async close()
```

create_collection

Creates a new collection in the data store.

Python

```
async create_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
collection_name Required*	

Returns

[+] Expand table

Type	Description
	None

delete_collection

Deletes a collection from the data store.

Python

```
async delete_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

does_collection_exist

Determines if a collection exists in the data store.

Python

```
async does_collection_exist(collection_name: str) -> bool
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	bool – True if given collection exists, False if not.

get

Gets a memory record from the data store. Does not guarantee that the collection exists.

Python

```
async get(collection_name: str, key: str, with_embedding: bool) ->
```

MemoryRecord

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
get. Required*	<xref:<xref:key {str} -- The unique id associated with the memory record to>>
true Required*	<xref:<xref:with_embedding {bool} -- If>>
record. Required*	<xref:<xref:the embedding will be returned in the memory>>
collection_name Required*	
key Required*	
with_embedding Required*	

Returns

[+] Expand table

Type	Description
	MemoryRecord – The memory record if found

get_batch

Gets a batch of memory records from the data store. Does not guarantee that the collection exists.

Python

```
async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
get. Required*	<xref:keys {List}>[str]<xref:-- The unique ids associated with the memory records to>
true Required*	<xref:<xref:with_embeddings {bool} -- If>>
records. Required*	<xref:<xref:the embedding will be returned in the memory>>
collection_name Required*	
keys Required*	
with_embeddings Required*	

Returns

[+] Expand table

Type	Description
	List[MemoryRecord] – The memory records associated with the unique keys provided.

get_collections

Gets all collection names in the data store.

Python

```
async get_collections() -> list[str]
```

Returns

[Expand table

Type	Description
	List[str] – A group of collection names.

get_nearest_match

Gets the nearest match to an embedding of type float. Does not guarantee that the collection exists.

Python

```
async get_nearest_match(collection_name: str, embedding: ndarray,  
with_embedding: bool, min_relevance_score: float | None = None) ->  
tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]
```

Parameters

[Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
with. Required*	<xref:<xref:embedding {ndarray} -- The embedding to compare the collection's embeddings>>
result. Required*	<xref:<xref:min_relevance_score {float} -- The minimum relevance threshold for returned>>
true Required*	<xref:<xref:with_embedding {bool} -- If>>
record. Required*	<xref:<xref:the embeddings will be returned in the memory>>
collection_name Required*	

Name	Description
embedding Required*	
with_embedding Required*	
min_relevance_score	default value: None

Returns

[] Expand table

Type	Description
	Tuple[MemoryRecord, float] – A tuple consisting of the MemoryRecord and the similarity score as a float.

get_nearest_matches

Gets the nearest matches to an embedding of type float. Does not guarantee that the collection exists.

Python

```
async get_nearest_matches(collection_name: str, embedding: ndarray,
limit: int, with_embeddings: bool, min_relevance_score: float | None =
None) -> list[tuple[semantic_kernel.memory.memory_record.MemoryRecord,
float]]
```

Parameters

[] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
with. Required*	<xref:<xref:embedding {ndarray} -- The embedding to compare the collection's embeddings>>
return Required*	<xref:<xref:limit {int} -- The maximum number> of <xref:similarity results to>>

Name	Description
1. Required*	<xref:<xref:defaults to>>
results. Required*	<xref:<xref:min_relevance_score {float} -- The minimum relevance threshold for returned>>
true Required*	<xref:<xref:with_embeddings {bool} -- If>>
records. Required*	<xref:<xref:the embeddings will be returned in the memory>>
collection_name Required*	
embedding Required*	
limit Required*	
with_embeddings Required*	
min_relevance_score	default value: None

Returns

[\[\] Expand table](#)

Type	Description
	List[Tuple[MemoryRecord, float]] – A list of tuples where item1 is a MemoryRecord and item2 is its similarity score as a float.

remove

Removes a memory record from the data store. Does not guarantee that the collection exists.

Python

```
async remove(collection_name: str, key: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
remove. Required*	<xref:<xref:key {str} -- The unique id associated with the memory record to>>
collection_name Required*	
key Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

remove_batch

Removes a batch of memory records from the data store. Does not guarantee that the collection exists.

Python

```
async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>

Name	Description
remove. Required*	<xref:keys {List}>[str]<xref:> -- The unique ids associated with the memory records to>
collection_name Required*	
keys Required*	

Returns

[] Expand table

Type	Description
	None

upsert

Upserts a memory record into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

Python

```
async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str}> -- The name associated with a collection of>>
upsert. Required*	<xref:<xref:record {MemoryRecord}> -- The memory record to>>
collection_name Required*	
record	

Name	Description
Required*	

Returns

[+] Expand table

Type	Description
	str – The unique identifier for the memory record.

upsert_batch

Upserts a group of memory records into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

Python

```
async upsert_batch(collection_name: str, records:
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
upsert. Required*	<xref:<xref:records {MemoryRecord} -- The memory records to>>
collection_name Required*	
records Required*	

Returns

[\[\] Expand table](#)

Type	Description
	List[str] – The unique identifiers for the memory records.

Attributes

database

database_name

index_name

num_candidates

is_experimental

Python

```
is_experimental = True
```

utils Module

Reference

Functions

document_to_memory_record

Converts a search result to a MemoryRecord.

Python

```
document_to_memory_record(data: dict, with_embeddings: bool) ->
MemoryRecord
```

Parameters

[\[\] Expand table](#)

Name	Description
data. Required*	<xref:<xref:dict -- Azure Cognitive Search result>>
data Required*	
with_embeddings Required*	

Returns

[\[\] Expand table](#)

Type	Description
	MemoryRecord – The MemoryRecord from Azure Cognitive Search Data Result.

memory_record_to_mongo_document

Convert a MemoryRecord to a dictionary

Python

```
memory_record_to_mongo_document(record: MemoryRecord) -> dict
```

Parameters

[+] Expand table

Name	Description
Result. Required*	<xref:<xref:record {MemoryRecord} -- The MemoryRecord from Azure Cognitive Search Data>>
record Required*	

Returns

[+] Expand table

Type	Description
	data {dict} – Dictionary data.

openai_plugin Package

Reference

Modules

[] [Expand table](#)

[openai_authentication_config](#)

[openai_function_execution_parameters](#)

[openai_utils](#)

Classes

[] [Expand table](#)

[OpenAIAuthenticationConfig](#)

OpenAI authentication configuration.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

[OpenAIFunctionExecutionParameters](#)

OpenAI function execution parameters.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

[OpenAIUtils](#)

Utility functions for OpenAI plugins.

Enums

[] [Expand table](#)

OpenAIAuthenticationType

openai_authentication_config Module

Reference

Classes

[] Expand table

OpenAIAuthenticationConfig	<p>OpenAI authentication configuration.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
--	--

Enums

[] Expand table

OpenAIAuthenticationType
OpenAIAuthorizationType

OpenAIAuthenticationConfig Class

Reference

OpenAI authentication configuration.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → OpenAIAuthenticationConfig

Constructor

Python

```
OpenAIAuthenticationConfig(*, type: OpenAIAuthenticationType | None = None,  
                           authorization_type: OpenAIAuthorizationType | None = None, client_url: Url |  
                           None = None, authorization_url: Url | None = None,  
                           authorization_content_type: str | None = None, scope: str | None = None,  
                           verification_tokens: dict[str, str] | None = None)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>type</code> Required*	
<code>authorization_type</code> Required*	
<code>client_url</code> Required*	
<code>authorization_url</code> Required*	
<code>authorization_content_type</code> Required*	

Name	Description
<code>scope</code> Required*	
<code>verification_tokens</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to `[ConfigDict]` [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo]` [`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =
{ 'authorization_content_type': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None), 'authorization_type':
FieldInfo(annotation=Union[OpenAIAuthorizationType, NoneType],
required=False, default=None), 'authorization_url':
```

```
FieldInfo(annotation=Union[Annotated[Url, UrlConstraints(max_length=2083, allowed_schemes=['http', 'https']), host_required=None, default_host=None, default_port=None, default_path=None]], NoneType], required=False, default=None), 'client_url': FieldInfo(annotation=Union[Annotated[Url, UrlConstraints(max_length=2083, allowed_schemes=['http', 'https']), host_required=None, default_host=None, default_port=None, default_path=None]], NoneType], required=False, default=None), 'scope': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'type': FieldInfo(annotation=Union[OpenAIAuthenticationType, NoneType], required=False, default=None), 'verification_tokens': FieldInfo(annotation=Union[dict[str, str], NoneType], required=False, default=None)}
```

authorization_content_type

Python

```
authorization_content_type: str | None
```

authorization_type

Python

```
authorization_type: OpenAIAuthorizationType | None
```

authorization_url

Python

```
authorization_url: Url | None
```

client_url

Python

```
client_url: Url | None
```

scope

Python

```
scope: str | None
```

type

```
Python
```

```
type: OpenAIAuthenticationType | None
```

verification_tokens

```
Python
```

```
verification_tokens: dict[str, str] | None
```

OpenAIAuthenticationType Enum

Reference

Inheritance builtins.str → OpenAIAuthenticationType
Enum → OpenAIAuthenticationType

Constructor

Python

```
OpenAIAuthenticationType(value, names=None, *, module=None, qualname=None,  
type=None, start=1, boundary=None)
```

Fields

[+] Expand table

NoneType
OAuth

OpenAIAuthorizationType Enum

Reference

Inheritance builtins.str → OpenAIAuthorizationType
Enum → OpenAIAuthorizationType

Constructor

Python

```
OpenAIAuthorizationType(value, names=None, *, module=None, qualname=None,  
type=None, start=1, boundary=None)
```

Fields

[] Expand table

Basic
Bearer

openai_function_execution_parameters Module

Reference

Classes

[] [Expand table](#)

[OpenAIFunctionExecutionParameters](#) OpenAI function execution parameters.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

OpenAIFunctionExecutionParameters Class

Reference

OpenAI function execution parameters.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [OpenAPIFunctionExecutionParameters](#) →
OpenAIFunctionExecutionParameters

Constructor

Python

```
OpenAIFunctionExecutionParameters(*, http_client: AsyncClient | None = None,  
auth_callback: Callable[..., Awaitable[Any]] | None = None,  
server_url_override: str | None = None, ignore_non_compliant_errors: bool =  
False, user_agent: str | None = None, enable_dynamic_payload: bool = True,  
enable_payload_namespacing: bool = False, operations_to_exclude: list[str] =  
None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>http_client</code> Required*	
<code>auth_callback</code> Required*	
<code>server_url_override</code> Required*	
<code>ignore_non_compliant_errors</code>	

Name	Description
Required*	
user_agent	
Required*	
enable_dynamic_payload	default value: True
enable_payload_namespacing	
Required*	
operations_to_exclude	
Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_callback':  
    FieldInfo(annotation=Union[Callable[..., Awaitable[Any]], NoneType],  
              required=False, default=None), 'enable_dynamic_payload':  
    FieldInfo(annotation=bool, required=False, default=True),  
    'enable_payload_namespacing': FieldInfo(annotation=bool, required=False,  
                                              default=False), 'http_client': FieldInfo(annotation=Union[AsyncClient,  
                                                       NoneType], required=False, default=None), 'ignore_non_compliant_errors':  
    FieldInfo(annotation=bool, required=False, default=False),  
    'operations_to_exclude': FieldInfo(annotation=list[str], required=False,  
                                         default_factory=list), 'server_url_override':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'user_agent': FieldInfo(annotation=Union[str, NoneType], required=False,  
                           default=None)}
```

auth_callback

Python

```
auth_callback: Callable[..., Awaitable[Any]] | None
```

enable_dynamic_payload

Python

```
enable_dynamic_payload: bool
```

enable_payload_namespacing

Python

```
enable_payload_namespacing: bool
```

http_client

Python

```
http_client: httpx.AsyncClient | None
```

ignore_non_compliant_errors

Python

```
ignore_non_compliant_errors: bool
```

operations_to_exclude

Python

```
operations_to_exclude: list[str]
```

server_url_override

Python

```
server_url_override: str | None
```

user_agent

Python

```
user_agent: str | None
```

openai_utils Module

Reference

Classes

[] [Expand table](#)

OpenAIUtils	Utility functions for OpenAI plugins.
-----------------------------	---------------------------------------

OpenAIUtils Class

Reference

Utility functions for OpenAI plugins.

Inheritance `builtins.object` → `OpenAIUtils`

Constructor

Python

```
OpenAIUtils()
```

Methods

[\[\] Expand table](#)

<code>parse_openai_manifest_for_openapi_spec_url</code>	Extract the OpenAPI Spec URL from the plugin JSON.
---	--

`parse_openai_manifest_for_openapi_spec_url`

Extract the OpenAPI Spec URL from the plugin JSON.

Python

```
static parse_openai_manifest_for_openapi_spec_url(plugin_json: dict[str, Any]) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>plugin_json</code> Required*	

OpenAIAuthenticationConfig Class

Reference

OpenAI authentication configuration.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → OpenAIAuthenticationConfig

Constructor

Python

```
OpenAIAuthenticationConfig(*, type: OpenAIAuthenticationType | None = None,  
                           authorization_type: OpenAIAuthorizationType | None = None, client_url: Url |  
                           None = None, authorization_url: Url | None = None,  
                           authorization_content_type: str | None = None, scope: str | None = None,  
                           verification_tokens: dict[str, str] | None = None)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>type</code> Required*	
<code>authorization_type</code> Required*	
<code>client_url</code> Required*	
<code>authorization_url</code> Required*	
<code>authorization_content_type</code> Required*	

Name	Description
<code>scope</code> Required*	
<code>verification_tokens</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to `[ConfigDict]` [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
    'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to `[FieldInfo]` [`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =
{ 'authorization_content_type': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None), 'authorization_type':
FieldInfo(annotation=Union[OpenAIAuthorizationType, NoneType],
required=False, default=None), 'authorization_url':
```

```
FieldInfo(annotation=Union[Annotated[Url, UrlConstraints(max_length=2083, allowed_schemes=['http', 'https']), host_required=None, default_host=None, default_port=None, default_path=None]], NoneType], required=False, default=None), 'client_url': FieldInfo(annotation=Union[Annotated[Url, UrlConstraints(max_length=2083, allowed_schemes=['http', 'https']), host_required=None, default_host=None, default_port=None, default_path=None]], NoneType], required=False, default=None), 'scope': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'type': FieldInfo(annotation=Union[OpenAIAuthenticationType, NoneType], required=False, default=None), 'verification_tokens': FieldInfo(annotation=Union[dict[str, str], NoneType], required=False, default=None)}
```

authorization_content_type

Python

```
authorization_content_type: str | None
```

authorization_type

Python

```
authorization_type: OpenAIAuthorizationType | None
```

authorization_url

Python

```
authorization_url: Url | None
```

client_url

Python

```
client_url: Url | None
```

scope

Python

```
scope: str | None
```

type

```
Python
```

```
type: OpenAIAuthenticationType | None
```

verification_tokens

```
Python
```

```
verification_tokens: dict[str, str] | None
```

OpenAIAuthenticationType Enum

Reference

Inheritance builtins.str → OpenAIAuthenticationType
Enum → OpenAIAuthenticationType

Constructor

Python

```
OpenAIAuthenticationType(value, names=None, *, module=None, qualname=None,  
type=None, start=1, boundary=None)
```

Fields

[+] Expand table

NoneType
OAuth

OpenAIFunctionExecutionParameters Class

Reference

OpenAI function execution parameters.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [OpenAPIFunctionExecutionParameters](#) →
OpenAIFunctionExecutionParameters

Constructor

Python

```
OpenAIFunctionExecutionParameters(*, http_client: AsyncClient | None = None,  
auth_callback: Callable[..., Awaitable[Any]] | None = None,  
server_url_override: str | None = None, ignore_non_compliant_errors: bool =  
False, user_agent: str | None = None, enable_dynamic_payload: bool = True,  
enable_payload_namespacing: bool = False, operations_to_exclude: list[str] =  
None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>http_client</code> Required*	
<code>auth_callback</code> Required*	
<code>server_url_override</code> Required*	
<code>ignore_non_compliant_errors</code>	

Name	Description
Required*	
user_agent	
Required*	
enable_dynamic_payload	default value: True
enable_payload_namespacing	
Required*	
operations_to_exclude	
Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_callback':  
    FieldInfo(annotation=Union[Callable[..., Awaitable[Any]], NoneType],  
              required=False, default=None), 'enable_dynamic_payload':  
    FieldInfo(annotation=bool, required=False, default=True),  
    'enable_payload_namespacing': FieldInfo(annotation=bool, required=False,  
                                              default=False), 'http_client': FieldInfo(annotation=Union[AsyncClient,  
                                                       NoneType], required=False, default=None), 'ignore_non_compliant_errors':  
    FieldInfo(annotation=bool, required=False, default=False),  
    'operations_to_exclude': FieldInfo(annotation=list[str], required=False,  
                                         default_factory=list), 'server_url_override':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'user_agent': FieldInfo(annotation=Union[str, NoneType], required=False,  
                           default=None)}
```

auth_callback

Python

```
auth_callback: Callable[..., Awaitable[Any]] | None
```

OpenAIUtils Class

Reference

Utility functions for OpenAI plugins.

Inheritance `builtins.object` → `OpenAIUtils`

Constructor

Python

```
OpenAIUtils()
```

Methods

[\[\] Expand table](#)

<code>parse_openai_manifest_for_openapi_spec_url</code>	Extract the OpenAPI Spec URL from the plugin JSON.
---	--

`parse_openai_manifest_for_openapi_spec_url`

Extract the OpenAPI Spec URL from the plugin JSON.

Python

```
static parse_openai_manifest_for_openapi_spec_url(plugin_json: dict[str, Any]) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>plugin_json</code> Required*	

openapi_plugin Package

Reference

Modules

[] Expand table

[openapi_function_execution_parameters](#)

[openapi_manager](#)

[openapi_parser](#)

[openapi_runner](#)

Classes

[] Expand table

[OpenAPIFunctionExecutionParameters](#) OpenAPI function execution parameters.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

rest_api_operation Module

Reference

Classes

[] [Expand table](#)

[RestApiOperation](#)

Note: This class is experimental and may change in the future.

RestApiOperation Class

Reference

Note: This class is experimental and may change in the future.

Inheritance builtins.object → RestApiOperation

Constructor

Python

```
RestApiOperation(id: str, method: str, server_url: str, path: str, summary: str | None = None, description: str | None = None, params: list[semantic_kernel.connectors.openapi_plugin.models.rest_api_operation_parameter.RestApiOperationParameter] | None = None, request_body: RestApiOperationPayload | None = None, responses: dict[str, semantic_kernel.connectors.openapi_plugin.models.rest_api_operation_expected_response.RestApiOperationExpectedResponse] | None = None)
```

Parameters

[] Expand table

Name	Description
id Required*	
method Required*	
server_url Required*	
path Required*	
summary	default value: None
description	default value: None
params	default value: None
request_body	default value: None

Name	Description
responses	default value: None

Methods

[+] Expand table

build_headers	
build_operation_url	
build_path	
build_query_string	
create_content_type_artificial_parameter	
create_payload_artificial_parameter	
get_default_response	
get_default_return_parameter	
get_parameters	
get_payload_parameters	
get_server_url	
replace_invalid_symbols	
url_join	Join a base URL and a path, correcting for any missing slashes.

build_headers

Python

```
build_headers(arguments: dict[str, Any]) -> dict[str, str]
```

Parameters

[+] Expand table

Name	Description
arguments Required*	

build_operation_url

Python

```
build_operation_url(arguments, server_url_override=None,
api_host_url=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
arguments Required*	
server_url_override	default value: None
api_host_url	default value: None

build_path

Python

```
build_path(path_template: str, arguments: dict[str, Any]) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
path_template Required*	
arguments Required*	

build_query_string

Python

```
build_query_string(arguments: dict[str, Any]) -> str
```

Parameters

[\[+\] Expand table](#)

Name	Description
arguments Required*	

create_content_type_artificial_parameter

Python

```
create_content_type_artificial_parameter() -> RestApiOperationParameter
```

create_payload_artificial_parameter

Python

```
create_payload_artificial_parameter(operation: RestApiOperation) ->  
RestApiOperationParameter
```

Parameters

[\[+\] Expand table](#)

Name	Description
operation Required*	

get_default_response

Python

```
get_default_response(responses: dict[str,  
semantic_kernel.connectors.openapi_plugin.models.rest_api_operation_expec  
ted_response.RestApiOperationExpectedResponse], preferred_responses:  
list[str]) -> RestApiOperationExpectedResponse | None
```

Parameters

[\[\] Expand table](#)

Name	Description
responses Required*	
preferred_responses Required*	

get_default_return_parameter

Python

```
get_default_return_parameter(preferred_responses: list[str] | None =  
None) -> KernelParameterMetadata
```

Parameters

[\[\] Expand table](#)

Name	Description
preferred_responses	default value: None

get_parameters

Python

```
get_parameters(operation: RestApiOperation,  
add_payload_params_from_metadata: bool = True, enable_payload_spacing:  
bool = False) ->  
list[semantic_kernel.connectors.openapi_plugin.models.rest_api_operation_  
parameter.RestApiOperationParameter]
```

Parameters

[+] Expand table

Name	Description
operation Required*	
add_payload_params_from_metadata	default value: True
enable_payload_spacing	default value: False

get_payload_parameters

Python

```
get_payload_parameters(operation: RestApiOperation,  
use_parameters_from_metadata: bool, enable_namespacing: bool)
```

Parameters

[+] Expand table

Name	Description
operation Required*	
use_parameters_from_metadata Required*	
enable_namespacing Required*	

get_server_url

Python

```
get_server_url(server_url_override=None, api_host_url=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>server_url_override</code>	default value: None
<code>api_host_url</code>	default value: None

replace_invalid_symbols

Python

```
replace_invalid_symbols(parameter_name)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>parameter_name</code> Required*	

url_join

Join a base URL and a path, correcting for any missing slashes.

Python

```
url_join(base_url: str, path: str)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>base_url</code> Required*	
<code>path</code> Required*	

Attributes

CONTENT_TYPE_ARGUMENT_NAME

Python

```
CONTENT_TYPE_ARGUMENT_NAME = 'content-type'
```

INVALID_SYMBOLS_REGEX

Python

```
INVALID_SYMBOLS_REGEX = re.compile('[^0-9A-Za-z_]+')
```

MEDIA_TYPE_TEXT_PLAIN

Python

```
MEDIA_TYPE_TEXT_PLAIN = 'text/plain'
```

PAYLOAD_ARGUMENT_NAME

Python

```
PAYLOAD_ARGUMENT_NAME = 'payload'
```

is_experimental

Python

```
is_experimental = True
```

rest_api_operation_expected_response

Module

Reference

Classes

[] [Expand table](#)

RestApiOperationExpectedResponse	Note: This class is experimental and may change in the future.
--	--

RestApiOperationExpectedResponse Class

Reference

Note: This class is experimental and may change in the future.

Inheritance builtins.object → RestApiOperationExpectedResponse

Constructor

Python

```
RestApiOperationExpectedResponse(description: str, media_type: str, schema: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
description Required*	
media_type Required*	
schema	default value: None

Attributes

is_experimental

Python

```
is_experimental = True
```

rest_api_operation_parameter Module

Reference

Classes

 [Expand table](#)

[RestApiOperationParameter](#)

Note: This class is experimental and may change in the future.

RestApiOperationParameter Class

Reference

Note: This class is experimental and may change in the future.

Inheritance builtins.object → RestApiOperationParameter

Constructor

Python

```
RestApiOperationParameter(name: str, type: str, location:  
    RestApiOperationParameterLocation, style: RestApiOperationParameterStyle |  
    None = None, alternative_name: str | None = None, description: str | None =  
    None, is_required: bool = False, default_value: Any | None = None, schema:  
    str | None = None, response: RestApiOperationExpectedResponse | None = None)
```

Parameters

[+] Expand table

Name	Description
name Required*	
type Required*	
location Required*	
style	default value: None
alternative_name	default value: None
description	default value: None
is_required	default value: False
default_value	default value: None
schema	default value: None
response	default value: None

Attributes

is_experimental

Python

```
is_experimental = True
```

rest_api_operation_parameter_location Module

Reference

Enums

 Expand table

RestApiOperationParameterLocation	The location of the REST API operation parameter.
---	---

Note: This class is experimental and may change in the future.

RestApiOperationParameterLocation Enum

Reference

The location of the REST API operation parameter.

Note: This class is experimental and may change in the future.

Inheritance [Enum](#) → RestApiOperationParameterLocation

Constructor

Python

```
RestApiOperationParameterLocation(value, names=None, *, module=None,  
qualname=None, type=None, start=1, boundary=None)
```

Fields

[] [Expand table](#)

BODY
COOKIE
HEADER
PATH
QUERY
is_experimental

rest_api_operation_parameter_style Module

Reference

Enums

[] [Expand table](#)

RestApiOperationParameterStyle	Note: This class is experimental and may change in the future.
--	--

RestApiOperationParameterStyle Enum

Reference

Note: This class is experimental and may change in the future.

Inheritance [Enum](#) → RestApiOperationParameterStyle

Constructor

Python

```
RestApiOperationParameterStyle(value, names=None, *, module=None,  
qualname=None, type=None, start=1, boundary=None)
```

Fields

[] [Expand table](#)

SIMPLE
is_experimental

rest_api_operation_payload Module

Reference

Classes

 [Expand table](#)

[RestApiOperationPayload](#)

Note: This class is experimental and may change in the future.

RestApiOperationPayload Class

Reference

Note: This class is experimental and may change in the future.

Inheritance `builtins.object` → `RestApiOperationPayload`

Constructor

Python

```
RestApiOperationPayload(media_type: str, properties:  
list[semantic_kernel.connectors.openapi_plugin.models.rest_api_operation_pay  
load_property.RestApiOperationPayloadProperty], description: str | None =  
None, schema: str | None = None)
```

Parameters

[+] Expand table

Name	Description
<code>media_type</code> Required*	
<code>properties</code> Required*	
<code>description</code>	default value: None
<code>schema</code>	default value: None

Attributes

`is_experimental`

Python

```
is_experimental = True
```

rest_api_operation_payload_property Module

Reference

Classes

[] [Expand table](#)

RestApiOperationPayloadProperty	Note: This class is experimental and may change in the future.
---	--

RestApiOperationPayloadProperty Class

Reference

Note: This class is experimental and may change in the future.

Inheritance `builtins.object` → `RestApiOperationPayloadProperty`

Constructor

Python

```
RestApiOperationPayloadProperty(name: str, type: str, properties:  
    RestApiOperationPayloadProperty, description: str | None = None,  
    is_required: bool = False, default_value: Any | None = None, schema: str |  
    None = None)
```

Parameters

[+] Expand table

Name	Description
<code>name</code> Required*	
<code>type</code> Required*	
<code>properties</code> Required*	
<code>description</code>	default value: None
<code>is_required</code>	default value: False
<code>default_value</code>	default value: None
<code>schema</code>	default value: None

Attributes

`is_experimental`

Python

```
is_experimental = True
```

rest_api_operation_run_options Module

Reference

Classes

 [Expand table](#)

[RestApiOperationRunOptions](#)

The options for running the REST API operation.

Note: This class is experimental and may change in the future.

RestApiOperationRunOptions Class

Reference

The options for running the REST API operation.

Note: This class is experimental and may change in the future.

Inheritance builtins.object → RestApiOperationRunOptions

Constructor

Python

```
RestApiOperationRunOptions(server_url_override=None, api_host_url=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
server_url_override	default value: None
api_host_url	default value: None

Attributes

is_experimental

Python

```
is_experimental = True
```

rest_api_uri Module

Reference

Classes

[\[\] Expand table](#)

Uri	The Uri class that represents the URI.
------------	--

Note: This class is experimental and may change in the future.

Uri Class

Reference

The Uri class that represents the URI.

Note: This class is experimental and may change in the future.

Inheritance `builtins.object` → Uri

Constructor

Python

```
Uri(uri)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>uri</code> Required*	

Methods

[\[\] Expand table](#)

```
get_left_part
```

get_left_part

Python

```
get_left_part()
```

Attributes

is_experimental

Python

```
is_experimental = True
```

openapi_function_execution_parameters Module

Reference

Classes

[] [Expand table](#)

[OpenAPIFunctionExecutionParameters](#) OpenAPI function execution parameters.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

OpenAPIFunctionExecutionParameters Class

Reference

OpenAPI function execution parameters.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → OpenAPIFunctionExecutionParameters

Constructor

Python

```
OpenAPIFunctionExecutionParameters(*, http_client: AsyncClient | None =  
    None, auth_callback: Callable[..., Awaitable[Any]] | None = None,  
    server_url_override: str | None = None, ignore_non_compliant_errors: bool =  
    False, user_agent: str | None = None, enable_dynamic_payload: bool = True,  
    enable_payload_namespacing: bool = False, operations_to_exclude: list[str] =  
    None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
http_client Required*	
auth_callback Required*	
server_url_override Required*	

Name	Description
<code>ignore_non_compliant_errors</code> Required*	
<code>user_agent</code> Required*	
<code>enable_dynamic_payload</code>	default value: True
<code>enable_payload_namespacing</code> Required*	
<code>operations_to_exclude</code> Required*	

Methods

[\[\] Expand table](#)

[model_post_init](#)

Python

```
model_post_init(_OpenAPIFunctionExecutionParameters__context: Any) ->
None
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>_OpenAPIFunctionExecutionParameters__context</code> Required*	

Attributes

[model_computed_fields](#)

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_callback':
FieldInfo(annotation=Union[Callable[..., Awaitable[Any]], NoneType],
required=False, default=None), 'enable_dynamic_payload':
FieldInfo(annotation=bool, required=False, default=True),
'enable_payload_namespacing': FieldInfo(annotation=bool, required=False,
default=False), 'http_client': FieldInfo(annotation=Union[AsyncClient,
NoneType], required=False, default=None), 'ignore_non_compliant_errors':
FieldInfo(annotation=bool, required=False, default=False),
'operations_to_exclude': FieldInfo(annotation=list[str], required=False,
default_factory=list), 'server_url_override':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'user_agent': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None)}
```

auth_callback

Python

```
auth_callback: Callable[..., Awaitable[Any]] | None
```

enable_dynamic_payload

Python

```
enable_dynamic_payload: bool
```

enable_payload_namespacing

Python

```
enable_payload_namespacing: bool
```

http_client

Python

```
http_client: AsyncClient | None
```

ignore_non_compliant_errors

Python

```
ignore_non_compliant_errors: bool
```

is_experimental

Python

```
is_experimental = True
```

operations_to_exclude

Python

```
operations_to_exclude: list[str]
```

server_url_override

Python

```
server_url_override: str | None
```

user_agent

Python

```
user_agent: str | None
```

openapi_manager Module

Reference

Functions

create_functions_from_openapi

Creates the functions from OpenAPI document.

Args: plugin_name: The name of the plugin openapi_document_path: The OpenAPI document path, it must be a file path to the spec. execution_settings: The execution settings

Returns: list[KernelFunctionFromMethod]: the operations as functions

Note: This function is experimental and may change in the future.

Python

```
create_functions_from_openapi(plugin_name: str, openapi_document_path: str, execution_settings: OpenAIFunctionExecutionParameters | OpenAPIFunctionExecutionParameters | None = None) -> list[semantic_kernel.functions.kernel_function_from_method.KernelFunctionFromMethod]
```

Parameters

[+] Expand table

Name	Description
plugin_name Required*	
openapi_document_path Required*	
execution_settings	default value: None

openapi_parser Module

Reference

Classes

[] [Expand table](#)

OpenApiParser	<p>NOTE: SK Python only supports the OpenAPI Spec >=3.0</p> <p>Import an OpenAPI file.</p> <p>Args: openapi_file: The path to the OpenAPI file which can be local or a URL.</p> <p>Returns: The parsed OpenAPI file</p> <p>param openapi_file: The path to the OpenAPI file which can be local or a URL.</p> <p>return: The parsed OpenAPI file</p> <p>Note: This class is experimental and may change in the future.</p>
-------------------------------	--

OpenApiParser Class

Reference

NOTE: SK Python only supports the OpenAPI Spec >=3.0

Import an OpenAPI file.

Args: openapi_file: The path to the OpenAPI file which can be local or a URL.

Returns: The parsed OpenAPI file

param openapi_file: The path to the OpenAPI file which can be local or a URL.

return: The parsed OpenAPI file

Note: This class is experimental and may change in the future.

Inheritance builtins.object → OpenApiParser

Constructor

Python

```
OpenApiParser()
```

Methods

[+] Expand table

<code>create_rest_api_operations</code>	Create the REST API Operations from the parsed OpenAPI document.
---	--

<code>parse</code>	Parse the OpenAPI document.
--------------------	-----------------------------

create_rest_api_operations

Create the REST API Operations from the parsed OpenAPI document.

Python

```
create_rest_api_operations(parsed_document: Any, execution_settings:  
    OpenAIFunctionExecutionParameters | OpenAPIFunctionExecutionParameters |  
    None = None) -> dict[str,
```

```
semantic_kernel.connectors.openapi_plugin.models.rest_api_operation.RestA  
piOperation]
```

Parameters

[\[\] Expand table](#)

Name	Description
parsed_document Required*	The parsed OpenAPI document
execution_settings	The execution settings default value: None

Returns

[\[\] Expand table](#)

Type	Description
	A dictionary of RestApiOperation objects keyed by operationId

parse

Parse the OpenAPI document.

Python

```
parse(openapi_document: str) -> Any | dict[str, Any] | None
```

Parameters

[\[\] Expand table](#)

Name	Description
openapi_document Required*	

Attributes

PAYLOAD_PROPERTIES_HIERARCHY_MAX_DEPTH

Python

```
PAYLOAD_PROPERTIES_HIERARCHY_MAX_DEPTH = 10
```

is_experimental

Python

```
is_experimental = True
```

supported_media_types

Python

```
supported_media_types = ['application/json', 'text/plain']
```

openapi_runner Module

Reference

Classes

 [Expand table](#)

OpenApiRunner	The OpenApiRunner that runs the operations defined in the OpenAPI manifest
-------------------------------	--

Note: This class is experimental and may change in the future.

OpenApiRunner Class

Reference

The OpenApiRunner that runs the operations defined in the OpenAPI manifest

Note: This class is experimental and may change in the future.

Inheritance `builtins.object` → `OpenApiRunner`

Constructor

Python

```
OpenApiRunner(parsed_openapi_document: Mapping[str, str], auth_callback:  
Callable[[dict[str, str]], dict[str, str]] | None = None, http_client:  
AsyncClient | None = None, enable_dynamic_payload: bool = True,  
enable_payload_namespacing: bool = False)
```

Parameters

[] Expand table

Name	Description
<code>parsed_openapi_document</code> Required*	
<code>auth_callback</code>	default value: None
<code>http_client</code>	default value: None
<code>enable_dynamic_payload</code>	default value: True
<code>enable_payload_namespacing</code>	default value: False

Methods

[] Expand table

build_full_url	Build the full URL.
build_json_object	Build the JSON payload object.

build_json_payload	Build the JSON payload.
build_operation_payload	
build_operation_url	Build the operation URL.
get_argument_name_for_payload	
run_operation	

build_full_url

Build the full URL.

Python

```
build_full_url(base_url, query_string)
```

Parameters

[\[\] Expand table](#)

Name	Description
base_url Required*	
query_string Required*	

build_json_object

Build the JSON payload object.

Python

```
build_json_object(properties, arguments, property_namespace=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
properties Required*	
arguments Required*	
property_namespace	default value: None

build_json_payload

Build the JSON payload.

Python

```
build_json_payload(payload_metadata: RestApiOperationPayload, arguments: dict[str, Any]) -> tuple[str, str]
```

Parameters

[\[+\] Expand table](#)

Name	Description
payload_metadata Required*	
arguments Required*	

build_operation_payload

Python

```
build_operation_payload(operation: RestApiOperation, arguments: KernelArguments) -> tuple[str, str]
```

Parameters

[\[+\] Expand table](#)

Name	Description
operation Required*	
arguments Required*	

build_operation_url

Build the operation URL.

Python

```
build_operation_url(operation: RestApiOperation, arguments: KernelArguments, server_url_override=None, api_host_url=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
operation Required*	
arguments Required*	
server_url_override	default value: None
api_host_url	default value: None

get_argument_name_for_payload

Python

```
get_argument_name_for_payload(property_name, property_namespace=None)
```

Parameters

[\[\] Expand table](#)

Name	Description
property_name Required*	
property_namespace	default value: None

run_operation

Python

```
async run_operation(operation: RestApiOperation, arguments:
KernelArguments | None = None, options: RestApiOperationRunOptions | None
= None) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
operation Required*	
arguments	default value: None
options	default value: None

Attributes

is_experimental

Python

```
is_experimental = True
```

media_type_application_json

Python

```
media_type_application_json = 'application/json'
```

payload_argument_name

Python

```
payload_argument_name = 'payload'
```

OpenAPIFunctionExecutionParameters Class

Reference

OpenAPI function execution parameters.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → OpenAPIFunctionExecutionParameters

Constructor

Python

```
OpenAPIFunctionExecutionParameters(*, http_client: AsyncClient | None =  
    None, auth_callback: Callable[..., Awaitable[Any]] | None = None,  
    server_url_override: str | None = None, ignore_non_compliant_errors: bool =  
    False, user_agent: str | None = None, enable_dynamic_payload: bool = True,  
    enable_payload_namespacing: bool = False, operations_to_exclude: list[str] =  
    None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
http_client Required*	
auth_callback Required*	
server_url_override Required*	

Name	Description
<code>ignore_non_compliant_errors</code> Required*	
<code>user_agent</code> Required*	
<code>enable_dynamic_payload</code>	default value: True
<code>enable_payload_namespacing</code> Required*	
<code>operations_to_exclude</code> Required*	

Methods

[\[\] Expand table](#)

[model_post_init](#)

Python

```
model_post_init(_OpenAPIFunctionExecutionParameters__context: Any) ->
None
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>_OpenAPIFunctionExecutionParameters__context</code> Required*	

Attributes

[model_computed_fields](#)

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_callback':
FieldInfo(annotation=Union[Callable[..., Awaitable[Any]], NoneType],
required=False, default=None), 'enable_dynamic_payload':
FieldInfo(annotation=bool, required=False, default=True),
'enable_payload_namespacing': FieldInfo(annotation=bool, required=False,
default=False), 'http_client': FieldInfo(annotation=Union[AsyncClient,
NoneType], required=False, default=None), 'ignore_non_compliant_errors':
FieldInfo(annotation=bool, required=False, default=False),
'operations_to_exclude': FieldInfo(annotation=list[str], required=False,
default_factory=list), 'server_url_override':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'user_agent': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None)}
```

auth_callback

Python

```
auth_callback: Callable[..., Awaitable[Any]] | None
```

enable_dynamic_payload

Python

```
enable_dynamic_payload: bool
```

enable_payload_namespacing

Python

```
enable_payload_namespacing: bool
```

http_client

Python

```
http_client: AsyncClient | None
```

ignore_non_compliant_errors

Python

```
ignore_non_compliant_errors: bool
```

is_experimental

Python

```
is_experimental = True
```

operations_to_exclude

Python

```
operations_to_exclude: list[str]
```

server_url_override

Python

```
server_url_override: str | None
```

user_agent

Python

```
user_agent: str | None
```

search_engine Package

Reference

Modules

[+] [Expand table](#)

bing_connector
bing_connector_settings
connector
google_connector

Classes

[+] [Expand table](#)

BingConnector	A search engine connector that uses the Bing Search API to perform a web search Initializes a new instance of the BingConnector class.
GoogleConnector	A search engine connector that uses the Google Custom Search API to perform a web search.

bing_connector Module

Reference

Classes

 Expand table

BingConnector	A search engine connector that uses the Bing Search API to perform a web search
----------------------	---

Initializes a new instance of the `BingConnector` class.

BingConnector Class

Reference

A search engine connector that uses the Bing Search API to perform a web search

Initializes a new instance of the BingConnector class.

Inheritance [ConnectorBase](#) → [BingConnector](#)

Constructor

Python

```
BingConnector(api_key: str | None = None, env_file_path: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
None} Required*	<xref:<xref:env_file_path {str} > The Bing Search API key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional path to the .env file. If provided, the settings are read from this file path location.
api_key	default value: None
env_file_path	default value: None

Methods

[+] [Expand table](#)

search	Returns the search results of the query provided by pinging the Bing web search API. Returns <i>num_results</i> results and ignores the first <i>offset</i> .
------------------------	--

search

Returns the search results of the query provided by pinging the Bing web search API.
Returns *num_results* results and ignores the first *offset*.

Python

```
async search(query: str, num_results: int = 1, offset: int = 0) ->
list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
query Required*	search query
num_results	the number of search results to return default value: 1
offset	the number of search results to ignore default value: 0

Returns

[\[\] Expand table](#)

Type	Description
	list of search results

bing_connector_settings Module

Reference

Classes

[\[\] Expand table](#)

BingSettings Bing Connector settings

The settings are first loaded from environment variables with the prefix '`>>BING_<<`'. If the environment variables are not found, the settings can be loaded from a .env file with the encoding 'utf-8'. If the settings are not found in the .env file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '`>>BING_<<`' are:

- `api_key`: SecretStr - The Bing API key (Env var `BING_API_KEY`)

BingSettings Class

Reference

Bing Connector settings

The settings are first loaded from environment variables with the prefix '>>BING_<<'. If the environment variables are not found, the settings can be loaded from a .env file with the encoding 'utf-8'. If the settings are not found in the .env file, the settings are ignored; however, validation will fail alerting that the settings are missing.

Optional settings for prefix '>>BING_<<' are:

- `api_key`: SecretStr - The Bing API key (Env var BING_API_KEY)

Inheritance `pydantic_settings.main.BaseSettings` → `BingSettings`

Constructor

Python

```
BingSettings(_case_sensitive: bool | None = None, _env_prefix: str | None = None, _env_file: DotenvType | None = WindowsPath('.'), _env_file_encoding: str | None = None, _env_ignore_empty: bool | None = None, _env_nested_delimiter: str | None = None, _env_parse_none_str: str | None = None, _secrets_dir: str | Path | None = None, *, env_file_path: str | None = None, api_key: SecretStr | None = None)
```

Parameters

[+] Expand table

Name	Description
<code>_case_sensitive</code>	default value: None
<code>_env_prefix</code>	default value: None
<code>_env_file</code>	default value: .
<code>_env_file_encoding</code>	default value: None
<code>_env_ignore_empty</code>	default value: None

Name	Description
<code>_env_nested_delimiter</code>	default value: None
<code>_env_parse_none_str</code>	default value: None
<code>_secrets_dir</code>	default value: None

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>env_file_path</code> Required*	
<code>api_key</code> Required*	

Methods

[\[\] Expand table](#)

[create](#)

Python

```
create(**kwargs)
```

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[SettingsConfigDict] = {'arbitrary_types_allowed': True, 'case_sensitive': False, 'env_file': None, 'env_file_encoding': 'utf-8', 'env_ignore_empty': False, 'env_nested_delimiter': None, 'env_parse_none_str': None, 'env_prefix': 'BING_', 'extra': 'ignore', 'json_file': None, 'json_file_encoding': None, 'protected_namespaces': ('model_', 'settings_'), 'secrets_dir': None, 'toml_file': None, 'validate_default': True, 'yaml_file': None, 'yaml_file_encoding': None}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'api_key': FieldInfo(annotation=Union[SecretStr, NoneType], required=False, default=None), 'env_file_path': FieldInfo(annotation=Union[str, NoneType], required=False, default=None)}
```

api_key

Python

```
api_key: SecretStr | None
```

env_file_path

Python

```
env_file_path: str | None
```

Config Class

Reference

Inheritance builtins.object → Config

Constructor

Python

```
Config()
```

Attributes

case_sensitive

Python

```
case_sensitive = False
```

env_file

Python

```
env_file = None
```

env_file_encoding

Python

```
env_file_encoding = 'utf-8'
```

env_prefix

Python

```
env_prefix = 'BING_'
```

extra

Python

```
extra = 'ignore'
```

connector Module

Reference

Classes

 Expand table

ConnectorBase	Base class for search engine connectors
-------------------------------	---

ConnectorBase Class

Reference

Base class for search engine connectors

Inheritance [ABC](#) → ConnectorBase

Constructor

Python

```
ConnectorBase()
```

Methods

[\[\] Expand table](#)

[search](#)

Python

```
abstract async search(query: str, num_results: int = 1, offset: int = 0)  
-> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
query Required*	
num_results	default value: 1
offset	default value: 0

google_connector Module

Reference

Classes

[+] [Expand table](#)

GoogleConnector	A search engine connector that uses the Google Custom Search API to perform a web search.
---------------------------------	---

GoogleConnector Class

Reference

A search engine connector that uses the Google Custom Search API to perform a web search.

Inheritance [ConnectorBase](#) → GoogleConnector

Constructor

Python

```
GoogleConnector(api_key: str, search_engine_id: str)
```

Parameters

[\[+\] Expand table](#)

Name	Description
api_key Required*	
search_engine_id Required*	

Methods

[\[+\] Expand table](#)

search	Returns the search results of the query provided by pinging the Google Custom search API. Returns <i>num_results</i> results and ignores the first <i>offset</i> .
---------------	--

search

Returns the search results of the query provided by pinging the Google Custom search API. Returns *num_results* results and ignores the first *offset*.

Python

```
async search(query: str, num_results: int = 1, offset: int = 0) ->  
list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
query Required*	search query
num_results	the number of search results to return default value: 1
offset	the number of search results to ignore default value: 0

Returns

[\[\] Expand table](#)

Type	Description
	list of search results

BingConnector Class

Reference

A search engine connector that uses the Bing Search API to perform a web search

Initializes a new instance of the BingConnector class.

Inheritance [ConnectorBase](#) → [BingConnector](#)

Constructor

Python

```
BingConnector(api_key: str | None = None, env_file_path: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
None} Required*	<xref:<xref:env_file_path {str} > The Bing Search API key. If provided, will override the value in the env vars or .env file.
None} Required*	The optional path to the .env file. If provided, the settings are read from this file path location.
api_key	default value: None
env_file_path	default value: None

Methods

[+] [Expand table](#)

search	Returns the search results of the query provided by pinging the Bing web search API. Returns <i>num_results</i> results and ignores the first <i>offset</i> .
------------------------	--

search

Returns the search results of the query provided by pinging the Bing web search API.
Returns *num_results* results and ignores the first *offset*.

Python

```
async search(query: str, num_results: int = 1, offset: int = 0) ->
list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
query Required*	search query
num_results	the number of search results to return default value: 1
offset	the number of search results to ignore default value: 0

Returns

[\[\] Expand table](#)

Type	Description
	list of search results

GoogleConnector Class

Reference

A search engine connector that uses the Google Custom Search API to perform a web search.

Inheritance [ConnectorBase](#) → GoogleConnector

Constructor

Python

```
GoogleConnector(api_key: str, search_engine_id: str)
```

Parameters

[\[\] Expand table](#)

Name	Description
api_key Required*	
search_engine_id Required*	

Methods

[\[\] Expand table](#)

search	Returns the search results of the query provided by pinging the Google Custom search API. Returns <i>num_results</i> results and ignores the first <i>offset</i> .
---------------	--

search

Returns the search results of the query provided by pinging the Google Custom search API. Returns *num_results* results and ignores the first *offset*.

Python

```
async search(query: str, num_results: int = 1, offset: int = 0) ->  
list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
query Required*	search query
num_results	the number of search results to return default value: 1
offset	the number of search results to ignore default value: 0

Returns

[\[\] Expand table](#)

Type	Description
	list of search results

utils Package

Reference

Modules

[\[\] Expand table](#)

[document_loader](#)

Classes

[\[\] Expand table](#)

[DocumentLoader](#)

document_loader Module

Reference

Classes

[\[\] Expand table](#)

DocumentLoader

DocumentLoader Class

Reference

Inheritance builtins.object → DocumentLoader

Constructor

Python

```
DocumentLoader()
```

Methods

[+] Expand table

<code>from_uri</code>	Load the manifest from the given URL
-----------------------	--------------------------------------

from_uri

Load the manifest from the given URL

Python

```
async static from_uri(url: str, http_client: AsyncClient, auth_callback: Callable[[Any], None] | None, user_agent: str | None = 'Semantic-Kernel')
```

Parameters

[+] Expand table

Name	Description
<code>url</code> Required*	
<code>http_client</code> Required*	
<code>auth_callback</code> Required*	

Name	Description
user_agent	default value: Semantic-Kernel

DocumentLoader Class

Reference

Inheritance builtins.object → DocumentLoader

Constructor

Python

```
DocumentLoader()
```

Methods

[+] Expand table

<code>from_uri</code>	Load the manifest from the given URL
-----------------------	--------------------------------------

from_uri

Load the manifest from the given URL

Python

```
async static from_uri(url: str, http_client: AsyncClient, auth_callback: Callable[[Any], None] | None, user_agent: str | None = 'Semantic-Kernel')
```

Parameters

[+] Expand table

Name	Description
<code>url</code> Required*	
<code>http_client</code> Required*	
<code>auth_callback</code> Required*	

Name	Description
user_agent	default value: Semantic-Kernel

telemetry Module

Reference

Functions

prepend_semantic_kernel_to_user_agent

Prepend "Semantic-Kernel" to the User-Agent in the headers.

Python

```
prepend_semantic_kernel_to_user_agent(headers: dict[str, Any])
```

Parameters

[\[\] Expand table](#)

Name	Description
headers Required*	The existing headers dictionary.

Returns

[\[\] Expand table](#)

Type	Description
	The modified headers dictionary with "Semantic-Kernel" prepended to the User-Agent.

contents Package

Reference

Modules

[] Expand table

author_role
chat_history
chat_message_content
const
finish_reason
function_call_content
function_result_content
kernel_content
streaming_chat_message_content
streaming_content_mixin
streaming_text_content
text_content
types

Classes

[] Expand table

ChatHistory	This class holds the history of chat messages from a chat conversation. Note: the constructor takes a system_message parameter, which is not part of the class definition. This is to allow the system_message to be passed in as a keyword argument, but not be part of the class definition.
-----------------------------	---

Initializes a new instance of the ChatHistory class, optionally incorporating a message and/or a system message at the beginning of the chat history.

This constructor allows for flexible initialization with chat messages and an optional messages or a system message. If both 'messages' (a list of ChatMessageContent instances) and 'system_message' are provided, the 'system_message' is prepended to the list of messages, ensuring it appears as the first message in the history. If only 'system_message' is provided without any 'messages', the chat history is initialized with the 'system_message' as its first item. If 'messages' are provided without a 'system_message', the chat history is initialized with the provided messages as is.

Parameters:

- **<<data: Arbitrary keyword arguments. The constructor looks for two optional keys:
 - 'messages': Optional[List[ChatMessageContent]], a list of chat messages to include in the history.
 - 'system_message' Optional[str]: An optional string representing a system-generated message to be

included at the start of the chat history.

Note: The 'system_message' is not retained as part of the class's attributes; it's used during initialization and then discarded. The rest of the keyword arguments are passed to the superclass constructor and handled according to the Pydantic model's behavior.

ChatMessageContent

This is the class for chat message response content.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

FunctionCallContent

Class to hold a function call response.

Create a new model by parsing and validating input data from keyword arguments.

	<p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
FunctionResultContent	<p>This is the base class for text response content.</p> <p>All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
StreamingChatMessageContent	<p>This is the class for streaming chat message response content.</p> <p>All Chat Completion Services should return a instance of this class as streaming response, where each part of the response as it is streamed is converted to a instance of this class, the end-user will have to either do something directly or gather them and combine them into a new instance. A service can implement their own subclass of this class and return instances of that.</p> <p>All Chat Completion Services should return a instance of this class as response for streaming. Or they can implement their own subclass of this class and return an instance.</p>
StreamingTextContent	<p>This is the base class for streaming text response content.</p> <p>All Text Completion Services should return a instance of this class as streaming response. Or they can implement their own subclass of this class and return an instance.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
TextContent	<p>This is the base class for text response content.</p> <p>All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.</p>

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Enums

[] [Expand table](#)

AuthorRole	Author role enum
------------	------------------

author_role Module

Reference

Enums

[\[+\] Expand table](#)

AuthorRole	Author role enum
------------	------------------

AuthorRole Enum

Reference

Author role enum

Inheritance builtins.str → AuthorRole

[Enum](#) → AuthorRole

Constructor

Python

```
AuthorRole(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[+] [Expand table](#)

ASSISTANT
SYSTEM
TOOL
USER

chat_history Module

Reference

Classes

 Expand table

[ChatHistory](#) This class holds the history of chat messages from a chat conversation.

Note: the constructor takes a system_message parameter, which is not part of the class definition. This is to allow the system_message to be passed in as a keyword argument, but not be part of the class definition.

Initializes a new instance of the ChatHistory class, optionally incorporating a message and/or a system message at the beginning of the chat history.

This constructor allows for flexible initialization with chat messages and an optional messages or a system message. If both 'messages' (a list of ChatMessageContent instances) and 'system_message' are provided, the 'system_message' is prepended to the list of messages, ensuring it appears as the first message in the history. If only 'system_message' is provided without any 'messages', the chat history is initialized with the 'system_message' as its first item. If 'messages' are provided without a 'system_message', the chat history is initialized with the provided messages as is.

Parameters:

- `**<<data:` Arbitrary keyword arguments. The constructor looks for two optional keys:
 - 'messages': Optional[List[ChatMessageContent]], a list of chat messages to include in the history.
 - 'system_message' Optional[str]: An optional string representing a system-generated message to be

included at the start of the chat history.

Note: The 'system_message' is not retained as part of the class's attributes; it's used during initialization and then discarded. The rest of the keyword arguments are passed to the superclass constructor and handled according to the Pydantic model's behavior.

ChatHistory Class

Reference

This class holds the history of chat messages from a chat conversation.

Note: the constructor takes a `system_message` parameter, which is not part of the class definition. This is to allow the `system_message` to be passed in as a keyword argument, but not be part of the class definition.

Initializes a new instance of the `ChatHistory` class, optionally incorporating a message and/or a system message at the beginning of the chat history.

This constructor allows for flexible initialization with chat messages and an optional messages or a system message. If both 'messages' (a list of `ChatMessageContent` instances) and 'system_message' are provided, the 'system_message' is prepended to the list of messages, ensuring it appears as the first message in the history. If only 'system_message' is provided without any 'messages', the chat history is initialized with the 'system_message' as its first item. If 'messages' are provided without a 'system_message', the chat history is initialized with the provided messages as is.

Parameters:

- `**<<data`: Arbitrary keyword arguments. The constructor looks for two optional keys:
 - 'messages': `Optional[List[ChatMessageContent]]`, a list of chat messages to include in the history.
 - 'system_message' `Optional[str]`: An optional string representing a system-generated message to be

included at the start of the chat history.

Note: The 'system_message' is not retained as part of the class's attributes; it's used during initialization and then discarded. The rest of the keyword arguments are passed to the superclass constructor and handled according to the Pydantic model's behavior.

Inheritance [KernelBaseModel](#) → ChatHistory

Constructor

Python

```
ChatHistory(*, messages:  
list[semantic_kernel.contents.chat_message_content.ChatMessageContent])
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
messages Required*	

Methods

[\[\] Expand table](#)

add_assistant_message	Add an assistant message to the chat history.
add_assistant_message_list	
add_assistant_message_str	
add_message	Add a message to the history. This method accepts either a ChatMessageContent instance or a dictionary with the necessary information to construct a ChatMessageContent instance.
add_system_message	Add a system message to the chat history.
add_system_message_list	
add_system_message_str	
add_tool_message	Add a tool message to the chat history.
add_tool_message_list	
add_tool_message_str	
add_user_message	Add a user message to the chat history.
add_user_message_list	

add_user_message_str	
from_rendered_prompt	Create a ChatHistory instance from a rendered prompt.
load_chat_history_from_file	Loads the ChatHistory from a file.
remove_message	Remove a message from the history.
restore_chat_history	Restores a ChatHistory instance from a JSON string.
serialize	Serializes the ChatHistory instance to a JSON string.
store_chat_history_to_file	Stores the serialized ChatHistory to a file.
to_prompt	Return a string representation of the history.

add_assistant_message

Add an assistant message to the chat history.

Python

```
add_assistant_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_assistant_message_list

Python

```
add_assistant_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[] Expand table

Name	Description
content Required*	

add_assistant_message_str

Python

```
add_assistant_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
content Required*	

add_message

Add a message to the history.

This method accepts either a ChatMessageContent instance or a dictionary with the necessary information to construct a ChatMessageContent instance.

Python

```
add_message(message: ChatMessageContent | dict[str, Any], encoding: str | None = None, metadata: dict[str, Any] | None = None) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
message Required*	<xref:Union>[<xref:ChatMessageContent>, dict] The message to add, either as a pre-constructed ChatMessageContent instance or a dictionary specifying 'role' and 'content'.
encoding	<xref:Optional>[str] The encoding of the message. Required if 'message' is a dict. default value: None
metadata	<xref:Optional>[dict [str ,<xref: Any>]] Any metadata to attach to the message. Required if 'message' is a dict. default value: None

add_system_message

Add a system message to the chat history.

Python

```
add_system_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs) ->  
None
```

add_system_message_list

Python

```
add_system_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_system_message_str

Python

```
add_system_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_tool_message

Add a tool message to the chat history.

Python

```
add_tool_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_tool_message_list

Python

```
add_tool_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_tool_message_str

Python

```
add_tool_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_user_message

Add a user message to the chat history.

Python

```
add_user_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_user_message_list

Python

```
add_user_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_user_message_str

Python

```
add_user_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

from_rendered_prompt

Create a ChatHistory instance from a rendered prompt.

Python

```
from_rendered_prompt(rendered_prompt: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
rendered_prompt Required*	str The rendered prompt to convert to a ChatHistory instance.

Returns

[\[\] Expand table](#)

Type	Description
ChatHistory	The ChatHistory instance created from the rendered prompt.

load_chat_history_from_file

Loads the ChatHistory from a file.

Python

```
load_chat_history_from_file(file_path: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
file_path Required*	str The path to the file from which to load the ChatHistory.

Returns

[+] Expand table

Type	Description
ChatHistory	The deserialized ChatHistory instance.

remove_message

Remove a message from the history.

Python

```
remove_message(message: ChatMessageContent) -> bool
```

Parameters

[+] Expand table

Name	Description
message Required*	<xref:semantic_kernel.contents.chat_history.ChatMessageContent> The message to remove.

Returns

[+] Expand table

Type	Description
bool	True if the message was removed, False if the message was not found.

restore_chat_history

Restores a ChatHistory instance from a JSON string.

Python

```
restore_chat_history(chat_history_json: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
chat_history_json Required*	str The JSON string to deserialize into a ChatHistory instance.

Returns

[\[\] Expand table](#)

Type	Description
ChatHistory	The deserialized ChatHistory instance.

Exceptions

[\[\] Expand table](#)

Type	Description
ValueError	If the JSON string is invalid or the deserialized data fails validation.

serialize

Serializes the ChatHistory instance to a JSON string.

Python

```
serialize() -> str
```

Returns

[\[\] Expand table](#)

Type	Description
str	A JSON string representation of the ChatHistory instance.

Exceptions

[\[\] Expand table](#)

Type	Description
ValueError	If the ChatHistory instance cannot be serialized to JSON.

store_chat_history_to_file

Stores the serialized ChatHistory to a file.

Python

```
store_chat_history_to_file(file_path: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
file_path Required*	str The path to the file where the serialized data will be stored.

to_prompt

Return a string representation of the history.

Python

```
to_prompt() -> str
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'messages':  
FieldInfo(annotation=list[ChatMessageContent], required=True)}
```

messages

The list of chat messages in the history.

Python

```
messages:  
list[semantic_kernel.contents.chat_message_content.ChatMessageContent]
```

chat_message_content Module

Reference

Classes

 [Expand table](#)

ChatMessageContent This is the class for chat message response content.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

ChatMessageContent Class

Reference

This is the class for chat message response content.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Inheritance [KernelContent](#) → ChatMessageContent

Constructor

Python

```
ChatMessageContent(role: AuthorRole, items:  
list[Union[semantic_kernel.contents.text_content.TextContent,  
semantic_kernel.contents.streaming_text_content.StreamingTextContent,  
semantic_kernel.contents.function_result_content.FunctionResultContent,  
semantic_kernel.contents.function_call_content.FunctionCallContent]] | None  
= None, content: str | None = None, inner_content: Any | None = None, name:  
str | None = None, encoding: str | None = None, finish_reason: FinishReason  
| None = None, ai_model_id: str | None = None, metadata: dict[str, Any] |  
None = None)
```

Parameters

[+] Expand table

Name	Description
inner_content	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing. default value: None
ai_model_id	Optional[str] - The id of the AI model that generated this response. default value: None
metadata	Dict[str, Any] - Any metadata that should be attached to the response. default value: None

Name	Description
role Required*	ChatRole - The role of the chat message.
content	Optional[str] - The text of the response. default value: None
encoding	Optional[str] - The encoding of the text. default value: None
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata Required*	Dict[str, Any] - Any metadata that should be attached to the response.
role Required*	ChatRole - The role of the chat message.
content Required*	str - The text of the response.
items	list[TextContent, StreamingTextContent, FunctionCallContent, FunctionResultContent] - The content. default value: None
encoding Required*	Optional[str] - The encoding of the text.
name	default value: None
finish_reason	default value: None

Methods

[\[\] Expand table](#)

from_element	Create a new instance of ChatMessageContent from a XML element.
to_dict	Serialize the ChatMessageContent to a dictionary.
to_element	Convert the ChatMessageContent to an XML Element.
to_prompt	Convert the ChatMessageContent to a prompt.

from_element

Create a new instance of ChatMessageContent from a XML element.

Python

```
from_element(element: Element) -> ChatMessageContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element Required*	Element - The XML Element to create the ChatMessageContent from.

Returns

[\[\] Expand table](#)

Type	Description
	ChatMessageContent - The new instance of ChatMessageContent or a subclass.

to_dict

Serialize the ChatMessageContent to a dictionary.

Python

```
to_dict(role_key: str = 'role', content_key: str = 'content') ->
dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
role_key	default value: role

Name	Description
content_key	default value: content

Returns

[\[\] Expand table](#)

Type	Description
	dict - The dictionary representing the ChatMessageContent.

to_element

Convert the ChatMessageContent to an XML Element.

Python

```
to_element() -> Element
```

Parameters

[\[\] Expand table](#)

Name	Description
root_key Required*	str - The key to use for the root of the XML Element.

Returns

[\[\] Expand table](#)

Type	Description
	Element - The XML Element representing the ChatMessageContent.

to_prompt

Convert the ChatMessageContent to a prompt.

Python

```
to_prompt() -> str
```

Returns

[+] Expand table

Type	Description
	str - The prompt from the ChatMessageContent.

Attributes

content

Get the content of the response, will find the first TextContent's text.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'finish_reason': FieldInfo(annotation=Union[FinishReason,  
    NoneType], required=False, default=None), 'inner_content':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'items': FieldInfo(annotation=list[Union[TextContent,  
    StreamingTextContent, FunctionResultContent, FunctionCallContent]],  
    required=False, default_factory=list), 'metadata':  
    FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'role': FieldInfo(annotation=AuthorRole,  
    required=True)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

encoding

Python

```
encoding: str | None
```

finish_reason

Python

```
finish_reason: FinishReason | None
```

inner_content

Python

```
inner_content: Any | None
```

items

Python

```
items: list[Union[semantic_kernel.contents.text_content.TextContent,  
semantic_kernel.contents.streaming_text_content.StreamingTextContent,  
semantic_kernel.contents.function_result_content.FunctionResultContent,  
semantic_kernel.contents.function_call_content.FunctionCallContent]]
```

metadata

Python

```
metadata: dict[str, Any]
```

name

Python

```
name: str | None
```

role

Python

```
role: AuthorRole
```

const Module

Reference

finish_reason Module

Reference

Enums

 Expand table

FinishReason	Finish Reason enum
------------------------------	--------------------

FinishReason Enum

Reference

Finish Reason enum

Inheritance builtins.str → FinishReason

[Enum](#) → FinishReason

Constructor

Python

```
FinishReason(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[+] [Expand table](#)

CONTENT_FILTER
FUNCTION_CALL
LENGTH
STOP
TOOL_CALLS

function_call_content Module

Reference

Classes

[] Expand table

[FunctionCallContent](#) Class to hold a function call response.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FunctionCallContent Class

Reference

Class to hold a function call response.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelContent](#) → FunctionCallContent

Constructor

Python

```
FunctionCallContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, id: str | None, index: int | None = None, name: str | None = None, arguments: str | None = None)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>inner_content</code> Required*	
<code>ai_model_id</code> Required*	
<code>metadata</code> Required*	
<code>id</code> Required*	
<code>index</code> Required*	
<code>name</code>	

Name	Description
Required*	
arguments	
Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
parse_arguments	Parse the arguments into a dictionary.
split_name	Split the name into a plugin and function name.
split_name_dict	Split the name into a plugin and function name.
to_dict	Convert the instance to a dictionary.
to_element	Convert the function call to an Element.
to_kernel_arguments	Return the arguments as a KernelArguments instance.

from_element

Create an instance from an Element.

Python

```
from_element(element: Element) -> FunctionCallContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element	
Required*	

parse_arguments

Parse the arguments into a dictionary.

Python

```
parse_arguments() -> dict[str, Any] | None
```

split_name

Split the name into a plugin and function name.

Python

```
split_name() -> list[str]
```

split_name_dict

Split the name into a plugin and function name.

Python

```
split_name_dict() -> dict
```

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str | Any]
```

to_element

Convert the function call to an Element.

Python

```
to_element() -> Element
```

to_kernel_arguments

Return the arguments as a `KernelArguments` instance.

Python

```
to_kernel_arguments() -> KernelArguments
```

Attributes

`function_name`

Get the function name.

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [`FieldInfo`][`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'arguments': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'id': FieldInfo(annotation=Union[str, NoneType],  
    required=True), 'index': FieldInfo(annotation=Union[int, NoneType],  
    required=False, default=None), 'inner_content':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'metadata': FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None)}
```

plugin_name

Get the plugin name.

ai_model_id

Python

```
ai_model_id: str | None
```

arguments

Python

```
arguments: str | None
```

id

Python

```
id: str | None
```

index

Python

```
index: int | None
```

inner_content

Python

```
inner_content: Any | None
```

metadata

Python

```
metadata: dict[str, Any]
```

name

Python

```
name: str | None
```

function_result_content Module

Reference

Classes

 [Expand table](#)

FunctionResultContent This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FunctionResultContent Class

Reference

This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelContent](#) → FunctionResultContent

Constructor

Python

```
FunctionResultContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, id: str, name: str | None = None, result: str, encoding: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
inner_content Required*	Any - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	str None - The id of the AI model that generated this response.
metadata Required*	dict[str, Any] - Any metadata that should be attached to the response.
text Required*	str None - The text of the response.

Name	Description
encoding Required*	str None - The encoding of the text.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
id Required*	
name Required*	
result Required*	
encoding Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
from_function_call_content_and_result	Create an instance from a FunctionCallContent and a result.
split_name	Split the name into a plugin and function name.
to_chat_message_content	Convert the instance to a ChatMessageContent.
to_dict	Convert the instance to a dictionary.

[to_element](#)

Convert the instance to an Element.

from_element

Create an instance from an Element.

Python

```
from_element(element: Element) -> FunctionResultContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element Required*	

from_function_call_content_and_result

Create an instance from a FunctionCallContent and a result.

Python

```
from_function_call_content_and_result(function_call_content:
FunctionCallContent, result: FunctionResult | TextContent |
ChatMessageContent | Any, metadata: dict[str, Any] = {}) ->
FunctionResultContent
```

Parameters

[\[\] Expand table](#)

Name	Description
function_call_content Required*	
result Required*	

Name	Description
metadata	default value: {}

split_name

Split the name into a plugin and function name.

Python

```
split_name() -> list[str]
```

to_chat_message_content

Convert the instance to a ChatMessageContent.

Python

```
to_chat_message_content(unwrap: bool = False) -> ChatMessageContent
```

Parameters

[\[\] Expand table](#)

Name	Description
unwrap	default value: False

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str]
```

to_element

Convert the instance to an Element.

Python

```
to_element() -> Element
```

Attributes

function_name

Get the function name.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [[pydantic.config.ConfigDict](#)].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][[pydantic.fields.FieldInfo](#)].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None), 'id': FieldInfo(annotation=str, required=True),
'inner_content': FieldInfo(annotation=Union[Any, NoneType],
```

```
required=False, default=None), 'metadata': FieldInfo(annotation=dict[str,  
Any], required=False, default_factory=dict), 'name':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'result': FieldInfo(annotation=str, required=True)}
```

plugin_name

Get the plugin name.

ai_model_id

Python

```
ai_model_id: str | None
```

encoding

Python

```
encoding: str | None
```

id

Python

```
id: str
```

inner_content

Python

```
inner_content: Any | None
```

metadata

Python

```
metadata: dict[str, Any]
```

name

Python

```
name: str | None
```

result

Python

```
result: str
```

kernel_content Module

Reference

Classes

[\[\] Expand table](#)

KernelContent Base class for all kernel contents.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

KernelContent Class

Reference

Base class for all kernel contents.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `KernelContent`

`ABC` → `KernelContent`

Constructor

Python

```
KernelContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>inner_content</code> Required*	
<code>ai_model_id</code> Required*	
<code>metadata</code> Required*	

Methods

[+] Expand table

```
from_element
```

```
to_dict
```

```
to_element
```

from_element

Python

```
abstract classmethod from_element(element: Any) -> KernelContent
```

Parameters

[+] Expand table

Name	Description
element Required*	

to_dict

Python

```
abstract to_dict() -> dict[str, Any]
```

to_element

Python

```
abstract to_element() -> Any
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'inner_content': FieldInfo(annotation=Union[Any, NoneType],  
required=False, default=None), 'metadata': FieldInfo(annotation=dict[str,  
Any], required=False, default_factory=dict)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

inner_content

Python

```
inner_content: Any | None
```

metadata

Python

```
metadata: dict[str, Any]
```

streaming_chat_message_content

Module

Reference

Classes

[] [Expand table](#)

[StreamingChatMessageContent](#) This is the class for streaming chat message response content.

All Chat Completion Services should return a instance of this class as streaming response, where each part of the response as it is streamed is converted to a instance of this class, the end-user will have to either do something directly or gather them and combine them into a new instance. A service can implement their own subclass of this class and return instances of that.

All Chat Completion Services should return a instance of this class as response for streaming. Or they can implement their own subclass of this class and return an instance.

StreamingChatMessageContent Class

Reference

This is the class for streaming chat message response content.

All Chat Completion Services should return a instance of this class as streaming response, where each part of the response as it is streamed is converted to a instance of this class, the end-user will have to either do something directly or gather them and combine them into a new instance. A service can implement their own subclass of this class and return instances of that.

All Chat Completion Services should return a instance of this class as response for streaming. Or they can implement their own subclass of this class and return an instance.

Inheritance [ChatMessageContent](#) → StreamingChatMessageContent

[StreamingContentMixin](#) → StreamingChatMessageContent

Constructor

Python

```
StreamingChatMessageContent(role: AuthorRole, choice_index: int, items: list[Union[semantic_kernel.contents.streaming_text_content.StreamingTextContent, semantic_kernel.contents.function_call_content.FunctionCallContent, semantic_kernel.contents.function_result_content.FunctionResultContent]] | None = None, content: str | None = None, inner_content: Any | None = None, name: str | None = None, encoding: str | None = None, finish_reason: FinishReason | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] | None = None)
```

Parameters

[+] Expand table

Name	Description
choice_index Required*	int - The index of the choice that generated this response.
inner_content	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.

Name	Description
	default value: None
ai_model_id	Optional[str] - The id of the AI model that generated this response. default value: None
metadata	Dict[str, Any] - Any metadata that should be attached to the response. default value: None
role Required*	Optional[ChatRole] - The role of the chat message, defaults to ASSISTANT.
content	Optional[str] - The text of the response. default value: None
encoding	Optional[str] - The encoding of the text. default value: None
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata Required*	Dict[str, Any] - Any metadata that should be attached to the response.
role Required*	ChatRole - The role of the chat message.
content Required*	str - The text of the response.
items	list[TextContent, FunctionCallContent, FunctionResultContent] - The content. default value: None
encoding Required*	Optional[str] - The encoding of the text.
name	default value: None
finish_reason	default value: None

Methods

[Expand table](#)

to_element	Convert the StreamingChatMessageContent to an XML Element.
----------------------------	--

to_element

Convert the StreamingChatMessageContent to an XML Element.

Python

```
to_element() -> Element
```

Parameters

[\[\] Expand table](#)

Name	Description
root_key Required*	str - The key to use for the root of the XML Element.

Returns

[\[\] Expand table](#)

Type	Description
	Element - The XML Element representing the StreamingChatMessageContent.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'choice_index': FieldInfo(annotation=int, required=True), 'encoding':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'finish_reason': FieldInfo(annotation=Union[FinishReason, NoneType],  
required=False, default=None), 'inner_content':  
FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
'items': FieldInfo(annotation=list[Union[TextContent,  
StreamingTextContent, FunctionResultContent, FunctionCallContent]],  
required=False, default_factory=list), 'metadata':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],  
required=False, default=None), 'role': FieldInfo(annotation=AuthorRole,  
required=True)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

choice_index

Python

```
choice_index: int
```

encoding

Python

```
encoding: str | None
```

finish_reason

Python

```
finish_reason: FinishReason | None
```

inner_content

Python

```
inner_content: Any | None
```

items

Python

```
items: list[ITEM_TYPES]
```

metadata

Python

```
metadata: dict[str, Any]
```

name

Python

```
name: str | None
```

role

Python

role: AuthorRole

streaming_content_mixin Module

Reference

Classes

 [Expand table](#)

[StreamingContentMixin](#) Mixin class for all streaming kernel contents.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

StreamingContentMixin Class

Reference

Mixin class for all streaming kernel contents.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `StreamingContentMixin`

`ABC` → `StreamingContentMixin`

Constructor

Python

```
StreamingContentMixin(*, choice_index: int)
```

Keyword-Only Parameters

[] Expand table

Name	Description
<code>choice_index</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'choice_index':  
FieldInfo(annotation=int, required=True)}
```

choice_index

Python

```
choice_index: int
```

streaming_text_content Module

Reference

Classes

[+] Expand table

[StreamingTextContent](#) This is the base class for streaming text response content.

All Text Completion Services should return a instance of this class as streaming response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

StreamingTextContent Class

Reference

This is the base class for streaming text response content.

All Text Completion Services should return a instance of this class as streaming response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [StreamingContentMixin](#) → StreamingTextContent
[TextContent](#) → StreamingTextContent

Constructor

Python

```
StreamingTextContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, text: str, encoding: str | None = None, choice_index: int)
```

Parameters

[] Expand table

Name	Description
choice_index Required*	int - The index of the choice that generated this response.
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata	Dict[str, Any] - Any metadata that should be attached to the response.

Name	Description
Required*	
text Required*	Optional[str] - The text of the response.
encoding Required*	Optional[str] - The encoding of the text.

Keyword-Only Parameters

[Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
text Required*	
encoding Required*	
choice_index Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'choice_index': FieldInfo(annotation=int, required=True), 'encoding':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'inner_content': FieldInfo(annotation=Union[Any, NoneType],  
required=False, default=None), 'metadata': FieldInfo(annotation=dict[str,  
Any], required=False, default_factory=dict), 'text':  
FieldInfo(annotation=str, required=True)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

choice_index

Python

```
choice_index: int
```

encoding

Python

```
encoding: str | None
```

inner_content

Python

```
inner_content: Any | None
```

metadata

Python

```
metadata: dict[str, Any]
```

text

Python

```
text: str
```

text_content Module

Reference

Classes

[+] Expand table

TextContent This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

TextContent Class

Reference

This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelContent](#) → TextContent

Constructor

Python

```
TextContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, text: str, encoding: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	Any - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	str None - The id of the AI model that generated this response.
metadata Required*	dict[str, Any] - Any metadata that should be attached to the response.
text Required*	str None - The text of the response.

Name	Description
encoding Required*	str None - The encoding of the text.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
text Required*	
encoding Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
to_dict	Convert the instance to a dictionary.
to_element	Convert the instance to an Element.

[from_element](#)

Create an instance from an Element.

Python

```
from_element(element: Element) -> TextContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element Required*	

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str]
```

to_element

Convert the instance to an Element.

Python

```
to_element() -> Element
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=None), 'inner_content': FieldInfo(annotation=Union[Any,  
NoneType], required=False, default=None), 'metadata':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'text': FieldInfo(annotation=str, required=True)}
```

ai_model_id

Python

```
ai_model_id: str | None
```

encoding

Python

```
encoding: str | None
```

inner_content

Python

```
inner_content: Any | None
```

metadata

Python

```
metadata: dict[str, Any]
```

text

Python

```
text: str
```

types Module

Reference

AuthorRole Enum

Reference

Author role enum

Inheritance builtins.str → AuthorRole

[Enum](#) → AuthorRole

Constructor

Python

```
AuthorRole(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[+] [Expand table](#)

ASSISTANT
SYSTEM
TOOL
USER

ChatHistory Class

Reference

This class holds the history of chat messages from a chat conversation.

Note: the constructor takes a system_message parameter, which is not part of the class definition. This is to allow the system_message to be passed in as a keyword argument, but not be part of the class definition.

Initializes a new instance of the ChatHistory class, optionally incorporating a message and/or a system message at the beginning of the chat history.

This constructor allows for flexible initialization with chat messages and an optional messages or a system message. If both 'messages' (a list of ChatMessageContent instances) and 'system_message' are provided, the 'system_message' is prepended to the list of messages, ensuring it appears as the first message in the history. If only 'system_message' is provided without any 'messages', the chat history is initialized with the 'system_message' as its first item. If 'messages' are provided without a 'system_message', the chat history is initialized with the provided messages as is.

Parameters:

- `**<<data: Arbitrary keyword arguments. The constructor looks for two optional keys:`
 - 'messages': `Optional[List[ChatMessageContent]]`, a list of chat messages to include in the history.
 - 'system_message' `Optional[str]`: An optional string representing a system-generated message to be

`included at the start of the chat history.`

Note: The 'system_message' is not retained as part of the class's attributes; it's used during initialization and then discarded. The rest of the keyword arguments are passed to the superclass constructor and handled according to the Pydantic model's behavior.

Inheritance [KernelBaseModel](#) → ChatHistory

Constructor

Python

```
ChatHistory(*, messages:  
list[semantic_kernel.contents.chat_message_content.ChatMessageContent])
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
messages Required*	

Methods

[\[\] Expand table](#)

add_assistant_message	Add an assistant message to the chat history.
add_assistant_message_list	
add_assistant_message_str	
add_message	Add a message to the history. This method accepts either a ChatMessageContent instance or a dictionary with the necessary information to construct a ChatMessageContent instance.
add_system_message	Add a system message to the chat history.
add_system_message_list	
add_system_message_str	
add_tool_message	Add a tool message to the chat history.
add_tool_message_list	
add_tool_message_str	
add_user_message	Add a user message to the chat history.
add_user_message_list	

add_user_message_str	
from_rendered_prompt	Create a ChatHistory instance from a rendered prompt.
load_chat_history_from_file	Loads the ChatHistory from a file.
remove_message	Remove a message from the history.
restore_chat_history	Restores a ChatHistory instance from a JSON string.
serialize	Serializes the ChatHistory instance to a JSON string.
store_chat_history_to_file	Stores the serialized ChatHistory to a file.
to_prompt	Return a string representation of the history.

add_assistant_message

Add an assistant message to the chat history.

Python

```
add_assistant_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_assistant_message_list

Python

```
add_assistant_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[] Expand table

Name	Description
content Required*	

add_assistant_message_str

Python

```
add_assistant_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
content Required*	

add_message

Add a message to the history.

This method accepts either a ChatMessageContent instance or a dictionary with the necessary information to construct a ChatMessageContent instance.

Python

```
add_message(message: ChatMessageContent | dict[str, Any], encoding: str | None = None, metadata: dict[str, Any] | None = None) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
message Required*	<xref:Union>[<xref:ChatMessageContent>, dict] The message to add, either as a pre-constructed ChatMessageContent instance or a dictionary specifying 'role' and 'content'.
encoding	<xref:Optional>[str] The encoding of the message. Required if 'message' is a dict. default value: None
metadata	<xref:Optional>[dict [str ,<xref: Any>]] Any metadata to attach to the message. Required if 'message' is a dict. default value: None

add_system_message

Add a system message to the chat history.

Python

```
add_system_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs) ->  
None
```

add_system_message_list

Python

```
add_system_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_system_message_str

Python

```
add_system_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_tool_message

Add a tool message to the chat history.

Python

```
add_tool_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_tool_message_list

Python

```
add_tool_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_tool_message_str

Python

```
add_tool_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_user_message

Add a user message to the chat history.

Python

```
add_user_message(content: str |  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

add_user_message_list

Python

```
add_user_message_list(content:  
list[semantic_kernel.contents.kernel_content.KernelContent], **kwargs:  
Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

add_user_message_str

Python

```
add_user_message_str(content: str, **kwargs: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

from_rendered_prompt

Create a ChatHistory instance from a rendered prompt.

Python

```
from_rendered_prompt(rendered_prompt: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
rendered_prompt Required*	str The rendered prompt to convert to a ChatHistory instance.

Returns

[\[\] Expand table](#)

Type	Description
ChatHistory	The ChatHistory instance created from the rendered prompt.

load_chat_history_from_file

Loads the ChatHistory from a file.

Python

```
load_chat_history_from_file(file_path: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
file_path Required*	str The path to the file from which to load the ChatHistory.

Returns

[+] Expand table

Type	Description
ChatHistory	The deserialized ChatHistory instance.

remove_message

Remove a message from the history.

Python

```
remove_message(message: ChatMessageContent) -> bool
```

Parameters

[+] Expand table

Name	Description
message Required*	ChatMessageContent The message to remove.

Returns

[+] Expand table

Type	Description
bool	True if the message was removed, False if the message was not found.

restore_chat_history

Restores a ChatHistory instance from a JSON string.

Python

```
restore_chat_history(chat_history_json: str) -> ChatHistory
```

Parameters

[\[\] Expand table](#)

Name	Description
chat_history_json Required*	str The JSON string to deserialize into a ChatHistory instance.

Returns

[\[\] Expand table](#)

Type	Description
ChatHistory	The deserialized ChatHistory instance.

Exceptions

[\[\] Expand table](#)

Type	Description
ValueError	If the JSON string is invalid or the deserialized data fails validation.

serialize

Serializes the ChatHistory instance to a JSON string.

Python

```
serialize() -> str
```

Returns

[\[\] Expand table](#)

Type	Description
str	A JSON string representation of the ChatHistory instance.

Exceptions

[\[\] Expand table](#)

Type	Description
ValueError	If the ChatHistory instance cannot be serialized to JSON.

store_chat_history_to_file

Stores the serialized ChatHistory to a file.

Python

```
store_chat_history_to_file(file_path: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
file_path Required*	str The path to the file where the serialized data will be stored.

to_prompt

Return a string representation of the history.

Python

```
to_prompt() -> str
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'messages':  
FieldInfo(annotation=list[ChatMessageContent], required=True)}
```

messages

The list of chat messages in the history.

Python

```
messages:  
list[semantic_kernel.contents.chat_message_content.ChatMessageContent]
```

ChatMessageContent Class

Reference

This is the class for chat message response content.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

All Chat Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Inheritance [KernelContent](#) → ChatMessageContent

Constructor

Python

```
ChatMessageContent(role: AuthorRole, items:  
list[Union[semantic_kernel.contents.text_content.TextContent,  
semantic_kernel.contents.streaming_text_content.StreamingTextContent,  
semantic_kernel.contents.function_result_content.FunctionResultContent,  
semantic_kernel.contents.function_call_content.FunctionCallContent]] | None  
= None, content: str | None = None, inner_content: Any | None = None, name:  
str | None = None, encoding: str | None = None, finish_reason: FinishReason  
| None = None, ai_model_id: str | None = None, metadata: dict[str, Any] |  
None = None)
```

Parameters

[] Expand table

Name	Description
inner_content	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing. default value: None
ai_model_id	Optional[str] - The id of the AI model that generated this response. default value: None
metadata	Dict[str, Any] - Any metadata that should be attached to the response. default value: None

Name	Description
role Required*	ChatRole - The role of the chat message.
content	Optional[str] - The text of the response. default value: None
encoding	Optional[str] - The encoding of the text. default value: None
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata Required*	Dict[str, Any] - Any metadata that should be attached to the response.
role Required*	ChatRole - The role of the chat message.
content Required*	str - The text of the response.
items	list[TextContent, StreamingTextContent, FunctionCallContent, FunctionResultContent] - The content. default value: None
encoding Required*	Optional[str] - The encoding of the text.
name	default value: None
finish_reason	default value: None

Methods

[\[\] Expand table](#)

from_element	Create a new instance of ChatMessageContent from a XML element.
to_dict	Serialize the ChatMessageContent to a dictionary.
to_element	Convert the ChatMessageContent to an XML Element.
to_prompt	Convert the ChatMessageContent to a prompt.

from_element

Create a new instance of ChatMessageContent from a XML element.

Python

```
from_element(element: Element) -> ChatMessageContent
```

Parameters

[\[+\] Expand table](#)

Name	Description
element Required*	Element - The XML Element to create the ChatMessageContent from.

Returns

[\[+\] Expand table](#)

Type	Description
	ChatMessageContent - The new instance of ChatMessageContent or a subclass.

to_dict

Serialize the ChatMessageContent to a dictionary.

Python

```
to_dict(role_key: str = 'role', content_key: str = 'content') ->
dict[str, Any]
```

Parameters

[\[+\] Expand table](#)

Name	Description
role_key	default value: role

Name	Description
content_key	default value: content

Returns

[\[\] Expand table](#)

Type	Description
	dict - The dictionary representing the ChatMessageContent.

to_element

Convert the ChatMessageContent to an XML Element.

Python

```
to_element() -> Element
```

Parameters

[\[\] Expand table](#)

Name	Description
root_key Required*	str - The key to use for the root of the XML Element.

Returns

[\[\] Expand table](#)

Type	Description
	Element - The XML Element representing the ChatMessageContent.

to_prompt

Convert the ChatMessageContent to a prompt.

Python

```
to_prompt() -> str
```

Returns

[+] Expand table

Type	Description
	str - The prompt from the ChatMessageContent.

Attributes

content

Get the content of the response, will find the first TextContent's text.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'finish_reason': FieldInfo(annotation=Union[FinishReason,  
    NoneType], required=False, default=None), 'inner_content':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'items': FieldInfo(annotation=list[Union[TextContent,  
    StreamingTextContent, FunctionResultContent, FunctionCallContent]],  
    required=False, default_factory=list), 'metadata':  
    FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None), 'role': FieldInfo(annotation=AuthorRole,  
    required=True)}
```

encoding

Python

```
encoding: str | None
```

finish_reason

Python

```
finish_reason: FinishReason | None
```

items

Python

```
items: list[Union[semantic_kernel.contents.text_content.TextContent,  
    semantic_kernel.contents.streaming_text_content.StreamingTextContent,  
    semantic_kernel.contents.function_result_content.FunctionResultContent,  
    semantic_kernel.contents.function_call_content.FunctionCallContent]]
```

name

Python

```
name: str | None
```

role

Python

```
role: AuthorRole
```

FunctionCallContent Class

Reference

Class to hold a function call response.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelContent](#) → FunctionCallContent

Constructor

Python

```
FunctionCallContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, id: str | None, index: int | None = None, name: str | None = None, arguments: str | None = None)
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
<code>inner_content</code> Required*	
<code>ai_model_id</code> Required*	
<code>metadata</code> Required*	
<code>id</code> Required*	
<code>index</code> Required*	
<code>name</code>	

Name	Description
Required*	
arguments	
Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
parse_arguments	Parse the arguments into a dictionary.
split_name	Split the name into a plugin and function name.
split_name_dict	Split the name into a plugin and function name.
to_dict	Convert the instance to a dictionary.
to_element	Convert the function call to an Element.
to_kernel_arguments	Return the arguments as a KernelArguments instance.

from_element

Create an instance from an Element.

Python

```
from_element(element: Element) -> FunctionCallContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element	
Required*	

parse_arguments

Parse the arguments into a dictionary.

Python

```
parse_arguments() -> dict[str, Any] | None
```

split_name

Split the name into a plugin and function name.

Python

```
split_name() -> list[str]
```

split_name_dict

Split the name into a plugin and function name.

Python

```
split_name_dict() -> dict
```

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str | Any]
```

to_element

Convert the function call to an Element.

Python

```
to_element() -> Element
```

to_kernel_arguments

Return the arguments as a `KernelArguments` instance.

Python

```
to_kernel_arguments() -> KernelArguments
```

Attributes

function_name

Get the function name.

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [`FieldInfo`][`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'arguments': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'id': FieldInfo(annotation=Union[str, NoneType],  
    required=True), 'index': FieldInfo(annotation=Union[int, NoneType],  
    required=False, default=None), 'inner_content':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'metadata': FieldInfo(annotation=dict[str, Any], required=False,  
    default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default=None)}
```

plugin_name

Get the plugin name.

arguments

Python

```
arguments: str | None
```

id

Python

```
id: str | None
```

index

Python

```
index: int | None
```

name

Python

```
name: str | None
```

FunctionResultContent Class

Reference

This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelContent](#) → FunctionResultContent

Constructor

Python

```
FunctionResultContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, id: str, name: str | None = None, result: str, encoding: str | None = None)
```

Parameters

[+] [Expand table](#)

Name	Description
inner_content Required*	Any - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	str None - The id of the AI model that generated this response.
metadata Required*	dict[str, Any] - Any metadata that should be attached to the response.
text Required*	str None - The text of the response.

Name	Description
encoding Required*	str None - The encoding of the text.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
id Required*	
name Required*	
result Required*	
encoding Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
from_function_call_content_and_result	Create an instance from a FunctionCallContent and a result.
split_name	Split the name into a plugin and function name.
to_chat_message_content	Convert the instance to a ChatMessageContent.
to_dict	Convert the instance to a dictionary.

[to_element](#)

Convert the instance to an Element.

from_element

Create an instance from an Element.

Python

```
from_element(element: Element) -> FunctionResultContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element Required*	

from_function_call_content_and_result

Create an instance from a FunctionCallContent and a result.

Python

```
from_function_call_content_and_result(function_call_content:
FunctionCallContent, result: FunctionResult | TextContent |
ChatMessageContent | Any, metadata: dict[str, Any] = {}) ->
FunctionResultContent
```

Parameters

[\[\] Expand table](#)

Name	Description
function_call_content Required*	
result Required*	

Name	Description
metadata	default value: {}

split_name

Split the name into a plugin and function name.

Python

```
split_name() -> list[str]
```

to_chat_message_content

Convert the instance to a ChatMessageContent.

Python

```
to_chat_message_content(unwrap: bool = False) -> ChatMessageContent
```

Parameters

[\[\] Expand table](#)

Name	Description
unwrap	default value: False

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str]
```

to_element

Convert the instance to an Element.

Python

```
to_element() -> Element
```

Attributes

function_name

Get the function name.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [[pydantic.config.ConfigDict](#)].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][[pydantic.fields.FieldInfo](#)].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,
default=None), 'id': FieldInfo(annotation=str, required=True),
'inner_content': FieldInfo(annotation=Union[Any, NoneType],
```

```
required=False, default=None), 'metadata': FieldInfo(annotation=dict[str,  
Any], required=False, default_factory=dict), 'name':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'result': FieldInfo(annotation=str, required=True)}
```

plugin_name

Get the plugin name.

encoding

Python

```
encoding: str | None
```

id

Python

```
id: str
```

name

Python

```
name: str | None
```

result

Python

```
result: str
```

StreamingChatMessageContent Class

Reference

This is the class for streaming chat message response content.

All Chat Completion Services should return a instance of this class as streaming response, where each part of the response as it is streamed is converted to a instance of this class, the end-user will have to either do something directly or gather them and combine them into a new instance. A service can implement their own subclass of this class and return instances of that.

All Chat Completion Services should return a instance of this class as response for streaming. Or they can implement their own subclass of this class and return an instance.

Inheritance [ChatMessageContent](#) → [StreamingChatMessageContent](#)

[StreamingContentMixin](#) → [StreamingChatMessageContent](#)

Constructor

Python

```
StreamingChatMessageContent(role: AuthorRole, choice_index: int, items: list[Union[semantic_kernel.contents.streaming_text_content.StreamingTextContent, semantic_kernel.contents.function_call_content.FunctionCallContent, semantic_kernel.contents.function_result_content.FunctionResultContent]] | None = None, content: str | None = None, inner_content: Any | None = None, name: str | None = None, encoding: str | None = None, finish_reason: FinishReason | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] | None = None)
```

Parameters

[+] Expand table

Name	Description
choice_index Required*	int - The index of the choice that generated this response.
inner_content	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.

Name	Description
	default value: None
ai_model_id	Optional[str] - The id of the AI model that generated this response. default value: None
metadata	Dict[str, Any] - Any metadata that should be attached to the response. default value: None
role Required*	Optional[ChatRole] - The role of the chat message, defaults to ASSISTANT.
content	Optional[str] - The text of the response. default value: None
encoding	Optional[str] - The encoding of the text. default value: None
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata Required*	Dict[str, Any] - Any metadata that should be attached to the response.
role Required*	ChatRole - The role of the chat message.
content Required*	str - The text of the response.
items	list[TextContent, FunctionCallContent, FunctionResultContent] - The content. default value: None
encoding Required*	Optional[str] - The encoding of the text.
name	default value: None
finish_reason	default value: None

Methods

[Expand table](#)

to_element	Convert the StreamingChatMessageContent to an XML Element.
-------------------	--

to_element

Convert the StreamingChatMessageContent to an XML Element.

Python

```
to_element() -> Element
```

Parameters

[\[\] Expand table](#)

Name	Description
root_key Required*	str - The key to use for the root of the XML Element.

Returns

[\[\] Expand table](#)

Type	Description
	Element - The XML Element representing the StreamingChatMessageContent.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'choice_index': FieldInfo(annotation=int, required=True), 'encoding':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'finish_reason': FieldInfo(annotation=Union[FinishReason, NoneType],
required=False, default=None), 'inner_content':
FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),
'items': FieldInfo(annotation=list[Union[TextContent,
StreamingTextContent, FunctionResultContent, FunctionCallContent]],
required=False, default_factory=list), 'metadata':
FieldInfo(annotation=dict[str, Any], required=False,
default_factory=dict), 'name': FieldInfo(annotation=Union[str, NoneType],
required=False, default=None), 'role': FieldInfo(annotation=AuthorRole,
required=True)}
```

StreamingTextContent Class

Reference

This is the base class for streaming text response content.

All Text Completion Services should return a instance of this class as streaming response.
Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [StreamingContentMixin](#) → StreamingTextContent
[TextContent](#) → StreamingTextContent

Constructor

Python

```
StreamingTextContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, text: str, encoding: str | None = None, choice_index: int)
```

Parameters

[] Expand table

Name	Description
choice_index Required*	int - The index of the choice that generated this response.
inner_content Required*	Optional[Any] - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	Optional[str] - The id of the AI model that generated this response.
metadata	Dict[str, Any] - Any metadata that should be attached to the response.

Name	Description
Required*	
text Required*	Optional[str] - The text of the response.
encoding Required*	Optional[str] - The encoding of the text.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
text Required*	
encoding Required*	
choice_index Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'choice_index': FieldInfo(annotation=int, required=True), 'encoding':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'inner_content': FieldInfo(annotation=Union[Any, NoneType],  
required=False, default=None), 'metadata': FieldInfo(annotation=dict[str,  
Any], required=False, default_factory=dict), 'text':  
FieldInfo(annotation=str, required=True)}
```

TextContent Class

Reference

This is the base class for text response content.

All Text Completion Services should return a instance of this class as response. Or they can implement their own subclass of this class and return an instance.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelContent](#) → TextContent

Constructor

Python

```
TextContent(*, inner_content: Any | None = None, ai_model_id: str | None = None, metadata: dict[str, Any] = None, text: str, encoding: str | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	Any - The inner content of the response, this should hold all the information from the response so even when not creating a subclass a developer can leverage the full thing.
ai_model_id Required*	str None - The id of the AI model that generated this response.
metadata Required*	dict[str, Any] - Any metadata that should be attached to the response.
text Required*	str None - The text of the response.

Name	Description
encoding Required*	str None - The encoding of the text.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
inner_content Required*	
ai_model_id Required*	
metadata Required*	
text Required*	
encoding Required*	

Methods

[\[\] Expand table](#)

from_element	Create an instance from an Element.
to_dict	Convert the instance to a dictionary.
to_element	Convert the instance to an Element.

[from_element](#)

Create an instance from an Element.

Python

```
from_element(element: Element) -> TextContent
```

Parameters

[\[\] Expand table](#)

Name	Description
element Required*	

to_dict

Convert the instance to a dictionary.

Python

```
to_dict() -> dict[str, str]
```

to_element

Convert the instance to an Element.

Python

```
to_element() -> Element
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
'encoding': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=None), 'inner_content': FieldInfo(annotation=Union[Any,  
NoneType], required=False, default=None), 'metadata':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'text': FieldInfo(annotation=str, required=True)}
```

encoding

Python

```
encoding: str | None
```

text

Python

```
text: str
```

exceptions Package

Reference

Modules

[] Expand table

[content_exceptions](#)

[function_exceptions](#)

[kernel_exceptions](#)

[memory_connector_exceptions](#)

[planner_exceptions](#)

[service_exceptions](#)

[template_engine_exceptions](#)

content_exceptions Module

Reference

Classes

[] [Expand table](#)

ContentAdditionException
ContentException
ContentInitializationError
ContentSerializationError
FunctionCallInvalidArgumentsException
FunctionCallInvalidNameException

ContentAdditionException Class

Reference

Inheritance [ContentException](#) → ContentAdditionException

Constructor

Python

```
ContentAdditionException()
```

ContentException Class

Reference

Inheritance [KernelException](#) → ContentException

Constructor

Python

```
ContentException()
```

ContentInitializationError Class

Reference

Inheritance [ContentException](#) → ContentInitializationError

Constructor

Python

```
ContentInitializationError()
```

ContentSerializationError Class

Reference

Inheritance [ContentException](#) → ContentSerializationError

Constructor

Python

```
ContentSerializationError()
```

FunctionCallInvalidArgumentsException Class

Reference

Inheritance [ContentException](#) → FunctionCallInvalidArgumentsException

Constructor

Python

```
FunctionCallInvalidArgumentsException()
```

FunctionCallInvalidNameException Class

Reference

Inheritance [ContentException](#) → FunctionCallInvalidNameException

Constructor

Python

```
FunctionCallInvalidNameException()
```

function_exceptions Module

Reference

Classes

[] Expand table

FunctionException
FunctionExecutionException
FunctionInitializationError
FunctionInvalidNameError
FunctionInvalidParamNameError
FunctionNameNotUniqueError
FunctionResultError
FunctionSyntaxError
PluginInitializationError
PluginInvalidNameError
PromptRenderingException

FunctionException Class

Reference

Inheritance [KernelException](#) → FunctionException

Constructor

Python

```
FunctionException()
```

FunctionExecutionException Class

Reference

Inheritance [FunctionException](#) → FunctionExecutionException

Constructor

Python

```
FunctionExecutionException()
```

FunctionInitializationError Class

Reference

Inheritance [FunctionException](#) → FunctionInitializationError

Constructor

Python

```
FunctionInitializationError(message: str)
```

Parameters

[] [Expand table](#)

Name	Description
message Required*	

FunctionInvalidNameError Class

Reference

Inheritance [FunctionSyntaxError](#) → FunctionInvalidNameError

Constructor

Python

```
FunctionInvalidNameError()
```

FunctionInvalidParamNameError Class

Reference

Inheritance [FunctionSyntaxError](#) → FunctionInvalidParamNameError

Constructor

Python

```
FunctionInvalidParamNameError()
```

FunctionNameNotUniqueError Class

Reference

Inheritance [FunctionSyntaxError](#) → FunctionNameNotUniqueError

Constructor

Python

```
FunctionNameNotUniqueError()
```

FunctionResultError Class

Reference

Inheritance [FunctionException](#) → FunctionResultError

Constructor

Python

```
FunctionResultError()
```

FunctionSyntaxError Class

Reference

Inheritance [FunctionException](#) → FunctionSyntaxError

Constructor

Python

```
FunctionSyntaxError()
```

PluginInitializationError Class

Reference

Inheritance [FunctionException](#) → PluginInitializationError

Constructor

Python

```
PluginInitializationError()
```

PluginInvalidNameError Class

Reference

Inheritance [FunctionSyntaxError](#) → `PluginInvalidNameError`

Constructor

Python

```
PluginInvalidNameError()
```

PromptRenderingException Class

Reference

Inheritance [FunctionException](#) → PromptRenderingException

Constructor

Python

```
PromptRenderingException()
```

kernel_exceptions Module

Reference

Classes

[] Expand table

KernelException
KernelFunctionAlreadyExistsError
KernelFunctionNotFoundError
KernelInvokeException
KernelPluginInvalidConfigurationError
KernelPluginNotFoundError
KernelServiceNotFoundError
OperationCancelledException

KernelException Class

Reference

Inheritance builtins.Exception → KernelException

Constructor

Python

```
KernelException()
```

KernelFunctionAlreadyExistsError Class

Reference

Inheritance [KernelException](#) → KernelFunctionAlreadyExistsError

Constructor

Python

```
KernelFunctionAlreadyExistsError()
```

KernelFunctionNotFoundError Class

Reference

Inheritance [KernelException](#) → KernelFunctionNotFoundError

Constructor

Python

```
KernelFunctionNotFoundError()
```

KernellInvokeException Class

Reference

Inheritance [KernelException](#) → KernellInvokeException

Constructor

Python

```
KernelInvokeException()
```

KernelPluginInvalidConfigurationError Class

Reference

Inheritance [KernelException](#) → KernelPluginInvalidConfigurationError

Constructor

Python

```
KernelPluginInvalidConfigurationError()
```

KernelPluginNotFoundError Class

Reference

Inheritance [KernelException](#) → KernelPluginNotFoundError

Constructor

Python

```
KernelPluginNotFoundError()
```

KernelServiceNotFoundError Class

Reference

Inheritance [KernelException](#) → KernelServiceNotFoundError

Constructor

Python

```
KernelServiceNotFoundError()
```

OperationCancelledException Class

Reference

Inheritance [KernelException](#) → OperationCancelledException

Constructor

Python

```
OperationCancelledException()
```

memory_connector_exceptions Module

Reference

Classes

[] Expand table

[MemoryConnectorException](#)

[MemoryConnectorInitializationError](#)

[MemoryConnectorResourceNotFound](#)

MemoryConnectorException Class

Reference

Inheritance [KernelException](#) → MemoryConnectorException

Constructor

Python

```
MemoryConnectorException()
```

MemoryConnectorInitializationError Class

Reference

Inheritance [MemoryConnectorException](#) → MemoryConnectorInitializationError

Constructor

Python

```
MemoryConnectorInitializationError()
```

MemoryConnectorResourceNotFound Class

Reference

Inheritance [MemoryConnectorException](#) → MemoryConnectorResourceNotFound

Constructor

Python

```
MemoryConnectorResourceNotFound()
```

planner_exceptions Module

Reference

Classes

[] Expand table

PlannerCreatePlanError
PlannerException
PlannerExecutionException
PlannerInvalidConfigurationError
PlannerInvalidGoalError
PlannerInvalidPlanError

PlannerCreatePlanError Class

Reference

Inheritance [PlannerException](#) → PlannerCreatePlanError

Constructor

Python

```
PlannerCreatePlanError()
```

PlannerException Class

Reference

Inheritance [KernelException](#) → PlannerException

Constructor

Python

```
PlannerException()
```

PlannerExecutionException Class

Reference

Inheritance [PlannerException](#) → PlannerExecutionException

Constructor

Python

```
PlannerExecutionException()
```

PlannerInvalidConfigurationError Class

Reference

Inheritance [PlannerException](#) → PlannerInvalidConfigurationError

Constructor

Python

```
PlannerInvalidConfigurationError()
```

PlannerInvalidGoalError Class

Reference

Inheritance [PlannerException](#) → PlannerInvalidGoalError

Constructor

Python

```
PlannerInvalidGoalError()
```

PlannerInvalidPlanError Class

Reference

Inheritance [PlannerException](#) → PlannerInvalidPlanError

Constructor

Python

```
PlannerInvalidPlanError()
```

service_exceptions Module

Reference

Classes

[] Expand table

ServiceContentFilterException
ServiceException
ServiceInitializationError
ServiceInvalidAuthError
ServiceInvalidExecutionSettingsError
ServiceInvalidRequestError
ServiceInvalidResponseError
ServiceInvalidTypeError
ServiceResourceNotFoundError
ServiceResponseException

ServiceContentFilterException Class

Reference

Inheritance [ServiceResponseException](#) → ServiceContentFilterException

Constructor

Python

```
ServiceContentFilterException()
```

ServiceException Class

Reference

Inheritance [KernelException](#) → ServiceException

Constructor

Python

```
ServiceException()
```

ServiceInitializationError Class

Reference

Inheritance [ServiceException](#) → ServiceInitializationError

Constructor

Python

```
ServiceInitializationError()
```

ServiceInvalidAuthError Class

Reference

Inheritance [ServiceException](#) → ServiceInvalidAuthError

Constructor

Python

```
ServiceInvalidAuthError()
```

ServiceInvalidExecutionSettingsError Class

Reference

Inheritance [ServiceResponseException](#) → ServiceInvalidExecutionSettingsError

Constructor

Python

```
ServiceInvalidExecutionSettingsError()
```

ServiceInvalidRequestError Class

Reference

Inheritance [ServiceResponseException](#) → ServiceInvalidRequestError

Constructor

Python

```
ServiceInvalidRequestError()
```

ServiceErrorResponse Class

Reference

Inheritance [ServiceResponseException](#) → ServiceErrorResponse

Constructor

Python

```
ServiceErrorResponse()
```

ServiceInvalidTypeError Class

Reference

Inheritance [ServiceResponseException](#) → ServiceInvalidTypeError

Constructor

Python

```
ServiceInvalidTypeError()
```

ServiceResourceNotFoundError Class

Reference

Inheritance [ServiceException](#) → ServiceResourceNotFoundError

Constructor

Python

```
ServiceResourceNotFoundError()
```

ServiceResponseException Class

Reference

Inheritance [ServiceException](#) → ServiceResponseException

Constructor

Python

```
ServiceResponseException()
```

template_engine_exceptions Module

Reference

Classes

[] Expand table

BlockException
BlockRenderException
BlockSyntaxError
CodeBlockRenderException
CodeBlockSyntaxError
CodeBlockTokenError
FunctionIdBlockSyntaxError
HandlebarsTemplateRenderException
HandlebarsTemplateSyntaxError
Jinja2TemplateRenderException
Jinja2TemplateSyntaxError
NamedArgBlockSyntaxError
TemplateRenderException
TemplateSyntaxError
ValBlockSyntaxError
VarBlockRenderError
VarBlockSyntaxError

BlockException Class

Reference

Inheritance [KernelException](#) → BlockException

Constructor

Python

```
BlockException()
```

BlockRenderException Class

Reference

Inheritance [BlockException](#) → BlockRenderException

Constructor

Python

```
BlockRenderException()
```

BlockSyntaxError Class

Reference

Inheritance `BlockException` → `BlockSyntaxError`

Constructor

Python

```
BlockSyntaxError()
```

CodeBlockRenderException Class

Reference

Inheritance [BlockRenderException](#) → CodeBlockRenderException

Constructor

Python

```
CodeBlockRenderException()
```

CodeBlockSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → CodeBlockSyntaxError

Constructor

Python

```
CodeBlockSyntaxError()
```

CodeBlockTokenError Class

Reference

Inheritance [BlockException](#) → CodeBlockTokenError

Constructor

Python

```
CodeBlockTokenError()
```

FunctionIdBlockSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → FunctionIdBlockSyntaxError

Constructor

Python

```
FunctionIdBlockSyntaxError(content: str)
```

Parameters

[+] [Expand table](#)

Name	Description
content Required*	

HandlebarsTemplateRenderException Class

Reference

Inheritance [BlockRenderException](#) → HandlebarsTemplateRenderException

Constructor

Python

```
HandlebarsTemplateRenderException()
```

HandlebarsTemplateSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → HandlebarsTemplateSyntaxError

Constructor

Python

```
HandlebarsTemplateSyntaxError()
```

Jinja2TemplateRenderException Class

Reference

Inheritance [BlockRenderException](#) → [Jinja2TemplateRenderException](#)

Constructor

Python

```
Jinja2TemplateRenderException()
```

Jinja2TemplateSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → `Jinja2TemplateSyntaxError`

Constructor

Python

```
Jinja2TemplateSyntaxError()
```

NamedArgBlockSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → NamedArgBlockSyntaxError

Constructor

Python

```
NamedArgBlockSyntaxError(content: str)
```

Parameters

[+] [Expand table](#)

Name	Description
content Required*	

TemplateRenderException Class

Reference

Inheritance [BlockRenderException](#) → [TemplateRenderException](#)

Constructor

Python

```
TemplateRenderException()
```

TemplateSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → TemplateSyntaxError

Constructor

Python

```
TemplateSyntaxError()
```

ValBlockSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → ValBlockSyntaxError

Constructor

Python

```
ValBlockSyntaxError(content: str)
```

Parameters

[+] [Expand table](#)

Name	Description
content Required*	

VarBlockRenderError Class

Reference

Inheritance [BlockRenderException](#) → VarBlockRenderError

Constructor

Python

```
VarBlockRenderError()
```

VarBlockSyntaxError Class

Reference

Inheritance [BlockSyntaxError](#) → VarBlockSyntaxError

Constructor

Python

```
VarBlockSyntaxError(content: str)
```

Parameters

[+] [Expand table](#)

Name	Description
content Required*	

functions Package

Reference

Modules

[] Expand table

kernel_function
function_result
kernel_arguments
kernel_function
kernel_function_decorator
kernel_function_extension
kernel_function_from_method
kernel_function_from_prompt
kernel_function_metadata
kernel_parameter_metadata
kernel_plugin
prompt_rendering_result
types

Classes

[] Expand table

FunctionResult	The result of a function. Create a new model by parsing and validating input data from keyword arguments. Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model. <i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.
--------------------------------	---

KernelArguments	<p>Initializes a new instance of the KernelArguments class, this is a dict-like class with the additional field for the execution_settings.</p> <p>This class is derived from a dict, hence behaves the same way, just adds the execution_settings as a dict, with service_id and the settings.</p>
KernelFunction	<p>Semantic Kernel function.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
KernelFunctionFromMethod	<p>Semantic Kernel Function from a method.</p> <p>Initializes a new instance of the KernelFunctionFromMethod class</p>
KernelFunctionFromPrompt	<p>Semantic Kernel Function from a prompt.</p> <p>Initializes a new instance of the KernelFunctionFromPrompt class</p>
KernelFunctionMetadata	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
KernelParameterMetadata	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
KernelPlugin	<p>Represents a Kernel Plugin with functions.</p> <p>This class behaves mostly like a dictionary, with functions as values and their names as keys. When you add a function, through <code>.set</code> or <code>setitem</code>, the function is copied, the metadata is deep-copied and the name of the plugin is set in the metadata and added to the dict of functions. This is done in the same way as a normal dict, so a existing key will be overwritten.</p> <p>Class methods: <code>from_object(plugin_name: str, plugin_instance: Any dict[str, Any], description: str None = None)</code>: Create a plugin from a existing object, like a custom class with annotated functions.</p>

```
from_directory(plugin_name: str, parent_directory: str, description: str | None = None): Create a plugin from a directory, parsing: .py files, .yaml files and directories with skprompt.txt and config.json files.
```

```
from.openapi( plugin_name: str, openapi_document_path: str, execution_settings: OpenAPIFunctionExecutionParameters | None = None, description: str | None = None):
```

```
Create a plugin from an OpenAPI document.
```

```
from.openapi( plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None):
```

```
Create a plugin from the Open AI manifest.
```

```
Create a KernelPlugin
```

Functions

kernel_function

Decorator for kernel functions, can be used directly as `@kernel_function` or with parameters `@kernel_function(name='function', description='I am a function.')`.

This decorator is used to mark a function as a kernel function. It also provides metadata for the function. The name and description can be left empty, and then the function name and docstring will be used.

The parameters are parsed from the function signature, use `typing.Annotated` to provide a description for the parameter, in python 3.8, use `typing_extensions.Annotated`.

To parse the type, first it checks if the parameter is annotated, and get's the description from there. After that it checks recursively until it reaches the lowest level, and it combines the types into a single comma-separated string, a `forwardRef` is also supported. All of this is stored in `kernel_function_parameters`.

The return type and description are parsed from the function signature, and that is stored in `kernel_function_return_type`, `kernel_function_return_description` and `kernel_function_return_required`.

It also checks if the function is a streaming type (generator or iterable, async or not), and that is stored as a bool in `kernel_function_streaming`.

Python

```
kernel_function(func: Callable[..., object] | None = None, name: str | None = None, description: str | None = None) -> Callable[..., Any]
```

Parameters

[+] Expand table

Name	Description
<code>name</code>	<xref:<xref:semantic_kernel.functions.str None>> default value: None
<code>description</code>	<xref:<xref:semantic_kernel.functions.str None>> if not supplied, the function docstring will be used, can be None. default value: None
<code>func</code>	default value: None

kernel_function

Decorator for kernel functions, can be used directly as `@kernel_function` or with parameters `@kernel_function(name='function', description='I am a function.')`.

This decorator is used to mark a function as a kernel function. It also provides metadata for the function. The name and description can be left empty, and then the function name and docstring will be used.

The parameters are parsed from the function signature, use `typing.Annotated` to provide a description for the parameter, in python 3.8, use `typing_extensions.Annotated`.

To parse the type, first it checks if the parameter is annotated, and get's the description from there. After that it checks recursively until it reaches the lowest level, and it combines the types into a single comma-separated string, a `forwardRef` is also supported. All of this is stored in `kernel_function_parameters`.

The return type and description are parsed from the function signature, and that is stored in `kernel_function_return_type`, `kernel_function_return_description` and `kernel_function_return_required`.

It also checks if the function is a streaming type (generator or iterable, async or not), and that is stored as a bool in `kernel_function_streaming`.

Python

```
kernel_function(func: Callable[..., object] | None = None, name: str | None = None, description: str | None = None) -> Callable[..., Any]
```

Parameters

[] [Expand table](#)

Name	Description
name	<xref:<xref:semantic_kernel.functions.str None>> default value: None
description	<xref:<xref:semantic_kernel.functions.str None>> if not supplied, the function docstring will be used, can be None. default value: None
func	default value: None

function_result Module

Reference

Classes

[\[\] Expand table](#)

FunctionResult The result of a function.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FunctionResult Class

Reference

The result of a function.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → FunctionResult

Constructor

Python

```
FunctionResult(*, function: KernelFunctionMetadata, value: Any, metadata: dict[str, Any] = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
function Required*	<xref:semantic_kernel.functions.function_result.KernelFunctionMetadata> The metadata of the function that was invoked.
value Required*	Any The value of the result.
metadata Required*	<xref:Mapping>[str ,<xref: Any>] The metadata of the result.

Keyword-Only Parameters

[\[\]](#) Expand table

Name	Description
function Required*	
value Required*	
metadata Required*	

Methods

[] [Expand table](#)

get_inner_content	Get the inner content of the function result when that is a KernelContent or subclass of the first item of the value if it is a list.
-----------------------------------	---

get_inner_content

Get the inner content of the function result when that is a KernelContent or subclass of the first item of the value if it is a list.

Python

```
get_inner_content()
```

Parameters

[] [Expand table](#)

Name	Description
index	default value: 0

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'function':  
FieldInfo(annotation=KernelFunctionMetadata, required=True), 'metadata':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'value': FieldInfo(annotation=Any, required=True)}
```

function

Python

```
function: KernelFunctionMetadata
```

metadata

Python

```
metadata: dict[str, Any]
```

value

Python

```
value: Any
```

kernel_arguments Module

Reference

Classes

[] [Expand table](#)

KernelArguments Initializes a new instance of the KernelArguments class, this is a dict-like class with the additional field for the execution_settings.

This class is derived from a dict, hence behaves the same way, just adds the execution_settings as a dict, with service_id and the settings.

KernelArguments Class

Reference

Initializes a new instance of the KernelArguments class, this is a dict-like class with the additional field for the execution_settings.

This class is derived from a dict, hence behaves the same way, just adds the execution_settings as a dict, with service_id and the settings.

Inheritance builtins.dict → KernelArguments

Constructor

Python

```
KernelArguments(settings: PromptExecutionSettings |  
list[PromptExecutionSettings] | dict[str, PromptExecutionSettings] | None =  
None, **kwargs: Any)
```

Parameters

[] Expand table

Name	Description
settings	<xref:PromptExecutionSettings List>[<xref:PromptExecutionSettings>]<xref: None> The settings for the execution. If a list is given, make sure all items in the list have a unique service_id as that is used as the key for the dict. default value: None
**kwargs Required*	dict[str,<xref: Any>]

kernel_function Module

Reference

Classes

[\[\] Expand table](#)

KernelFunction Semantic Kernel function.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

KernelFunction Class

Reference

Semantic Kernel function.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `KernelFunction`

Constructor

Python

```
KernelFunction(*, metadata: KernelFunctionMetadata)
```

Keyword-Only Parameters

[] Expand table

Name	Description
<code>metadata</code> Required*	

Methods

[] Expand table

<code>from_method</code>	Create a new instance of the <code>KernelFunctionFromMethod</code> class.
<code>from_prompt</code>	Create a new instance of the <code>KernelFunctionFromPrompt</code> class.
<code>function_copy</code>	Copy the function, can also override the <code>plugin_name</code> .
<code>invoke</code>	Invoke the function with the given arguments.

`invoke_stream` Invoke a stream async function with the given arguments.

from_method

Create a new instance of the KernelFunctionFromMethod class.

Python

```
from_method(method: Callable[..., Any], plugin_name: str | None = None,
            stream_method: Callable[..., Any] | None = None) ->
    KernelFunctionFromMethod
```

Parameters

[\[\] Expand table](#)

Name	Description
method Required*	
plugin_name	default value: None
stream_method	default value: None

from_prompt

Create a new instance of the KernelFunctionFromPrompt class.

Python

```
from_prompt(function_name: str, plugin_name: str, description: str | None
            = None, prompt: str | None = None, template_format: Literal['semantic-
            kernel', 'handlebars', 'jinja2'] = 'semantic-kernel', prompt_template:
            PromptTemplateBase | None = None, prompt_template_config:
            PromptTemplateConfig | None = None, prompt_execution_settings:
            PromptExecutionSettings | list[PromptExecutionSettings] | dict[str,
            PromptExecutionSettings] | None = None) -> KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
function_name Required*	
plugin_name Required*	
description	default value: None
prompt	default value: None
template_format	default value: semantic-kernel
prompt_template	default value: None
prompt_template_config	default value: None
prompt_execution_settings	default value: None

function_copy

Copy the function, can also override the plugin_name.

Python

```
function_copy(plugin_name: str | None = None) -> KernelFunction
```

Parameters

[\[\] Expand table](#)

Name	Description
plugin_name	str The new plugin name. default value: None

Returns

[\[\] Expand table](#)

Type	Description
KernelFunction	The copied function.

invoke

Invoke the function with the given arguments.

Python

```
async invoke(kernel: Kernel, arguments: KernelArguments | None = None,
metadata: dict[str, Any] = {}, **kwargs: Any) -> FunctionResult | None
```

Parameters

[\[+\] Expand table](#)

Name	Description
kernel Required*	<xref:semantic_kernel.functions.kernel_function.Kernel> The kernel
arguments	<xref:semantic_kernel.functions.kernel_function.KernelArguments> The Kernel arguments default value: None
kwargs Required*	Any Additional keyword arguments that will be added to the KernelArguments.
metadata	default value: {}

Returns

[\[+\] Expand table](#)

Type	Description
FunctionResult	The result of the function

invoke_stream

Invoke a stream async function with the given arguments.

Python

```
async invoke_stream(kernel: Kernel, arguments: KernelArguments | None = None,
metadata: dict[str, Any] = {}, **kwargs: Any) ->
AsyncGenerator[FunctionResult | list[StreamingContentMixin | Any], Any]
```

Parameters

[] Expand table

Name	Description
kernel Required*	<xref:semantic_kernel.functions.kernel_function.Kernel> The kernel
arguments	<xref:semantic_kernel.functions.kernel_function.KernelArguments> The Kernel arguments default value: None
kwargs Required*	Any Additional keyword arguments that will be added to the KernelArguments.
metadata	default value: {}

Attributes

description

The description of the function.

fully_qualified_name

is_prompt

Whether the function is semantic.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':  
FieldInfo(annotation=KernelFunctionMetadata, required=True)}
```

name

The name of the function. Must be upper/lower case letters and underscores with a minimum length of 1.

parameters

The parameters for the function.

plugin_name

The name of the plugin that contains this function. Must be upper/lower case letters and underscores with a minimum length of 1.

return_parameter

The return parameter for the function.

stream_function

The stream function for the function.

function

The function to call.

prompt_execution_settings

The AI prompt execution settings.

prompt_template_config

The prompt template configuration.

metadata

The metadata for the function.

Python

```
metadata: KernelFunctionMetadata
```

kernel_function_decorator Module

Reference

Functions

kernel_function

Decorator for kernel functions, can be used directly as @kernel_function or with parameters @kernel_function(name='function', description='I am a function.').

This decorator is used to mark a function as a kernel function. It also provides metadata for the function. The name and description can be left empty, and then the function name and docstring will be used.

The parameters are parsed from the function signature, use typing.Annotated to provide a description for the parameter, in python 3.8, use typing_extensions.Annotated.

To parse the type, first it checks if the parameter is annotated, and get's the description from there. After that it checks recursively until it reaches the lowest level, and it combines the types into a single comma-separated string, a forwardRef is also supported. All of this is stored in `kernel_function_parameters`.

The return type and description are parsed from the function signature, and that is stored in `kernel_function_return_type`, `kernel_function_return_description` and `kernel_function_return_required`.

It also checks if the function is a streaming type (generator or iterable, async or not), and that is stored as a bool in `kernel_function_streaming`.

Python

```
kernel_function(func: Callable[..., object] | None = None, name: str | None = None, description: str | None = None) -> Callable[..., Any]
```

Parameters

 Expand table

Name	Description
name	<xref:<xref:semantic_kernel.functions.kernel_function_decorator.str None>> default value: None
description	<xref:<xref:semantic_kernel.functions.kernel_function_decorator.str None>> if not supplied, the function docstring will be used, can be None. default value: None
func	default value: None

kernel_function_extension Module

Reference

Classes

 [Expand table](#)

KernelFunctionExtension	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
--------------------------------	--

KernelFunctionExtension Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → KernelFunctionExtension

[ABC](#) → KernelFunctionExtension

Constructor

Python

```
KernelFunctionExtension(*, plugins: dict[str,  
semantic_kernel.functions.kernel_plugin.KernelPlugin] = None)
```

Keyword-Only Parameters

[] Expand table

Name	Description
<code>plugins</code> Required*	

Methods

[] Expand table

add_function	Adds a function to the specified plugin.
add_functions	Adds a list of functions to the specified plugin.
add_plugin	Adds a plugin to the kernel's collection of plugins. If a plugin is provided, it uses that instance instead of creating a new KernelPlugin. See KernelPlugin.from_directory for more details on how the directory is parsed.
add_plugin_from_openai	Add a plugin from an OpenAPI document.
add_plugin_from_openapi	Add a plugin from the Open AI manifest.
add_plugins	Adds a list of plugins to the kernel's collection of plugins.
get_full_list_of_function_metadata	Get a list of all function metadata in the plugins.
get_function	Get a function by plugin_name and function_name.

<code>get_function_from_fully_qualified_function_name</code>	Get a function by its fully qualified name (<plugin_name>-<function_name>).
<code>get_list_of_function_metadata</code>	Get a list of all function metadata in the plugin collection.
<code>get_list_of_function_metadata_bool</code>	Get a list of the function metadata in the plugin collection
<code>get_list_of_function_metadata_filters</code>	Get a list of Kernel Function Metadata based on filters.
<code>get_plugin</code>	Get a plugin by name.
<code>rewrite_plugins</code>	Rewrite plugins to a dictionary.

add_function

Adds a function to the specified plugin.

Python

```
add_function(plugin_name: str, function: KERNEL_FUNCTION_TYPE | None = None, function_name: str | None = None, description: str | None = None, prompt: str | None = None, prompt_template_config: PromptTemplateConfig | None = None, prompt_execution_settings: PromptExecutionSettings | list[semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings] | dict[str, semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings] | None = None, template_format: Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel', prompt_template: PromptTemplateBase | None = None, return_plugin: bool = False, **kwargs: Any) -> KernelFunction | KernelPlugin
```

Parameters

[] Expand table

Name	Description
plugin_name Required*	str The name of the plugin to add the function to
function	<xref:KernelFunction Callable>[<xref:>, <xref: Any>] The function to add default value: None
function_name	str The name of the function default value: None
plugin_name Required*	The name of the plugin
description	<xref:><xref:semantic_kernel.functions.kernel_function_extension.str None>> The description of the function default value: None
prompt	<xref:><xref:semantic_kernel.functions.kernel_function_extension.str None>> The prompt template. default value: None

Name	Description
prompt_template_config	<xref: <xref:semantic_kernel.functions.kernel_function_extension.PromptTemplateConfig None>> The prompt template configuration default value: None
list[PromptExecutionSettings] Required*	PromptExecutionSettings] (<xref:prompt_execution_settings> (PromptExecutionSettings dict[str, PromptExecutionSettings] None): The execution settings, will be parsed into a dict.
template_format	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The format of the prompt template default value: semantic-kernel
prompt_template	<xref: <xref:semantic_kernel.functions.kernel_function_extension.PromptTemplateBase None>> The prompt template default value: None
return_plugin	bool If True, the plugin is returned instead of the function default value: False
kwargs Required*	Any Additional arguments
prompt_execution_settings	default value: None

Returns

[\[\]](#) Expand table

Type	Description
KernelFunction KernelPlugin	The function that was added, or the plugin if return_plugin is True

add_functions

Adds a list of functions to the specified plugin.

Python

```
add_functions(plugin_name: str, functions: list[KERNEL_FUNCTION_TYPE] | dict[str,  
KERNEL_FUNCTION_TYPE]) -> KernelPlugin
```

Parameters

[\[\]](#) Expand table

Name	Description
plugin_name Required*	str The name of the plugin to add the functions to
functions Required*	list[<xref:KernelFunction>]<xref: dict>[str,<xref: KernelFunction>] The functions to add

Returns

[Expand table](#)

Type	Description
KernelPlugin	The plugin that the functions were added to.

add_plugin

Adds a plugin to the kernel's collection of plugins. If a plugin is provided, it uses that instance instead of creating a new KernelPlugin. See `KernelPlugin.from_directory` for more details on how the directory is parsed.

Python

```
add_plugin(plugin: KernelPlugin | object | dict[str, Any] | None = None, plugin_name: str | None = None, parent_directory: str | None = None, description: str | None = None, class_init_arguments: dict[str, dict[str, Any]] | None = None) -> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin	<xref:KernelPlugin Any dict>[str,<xref: Any>] The plugin to add. This can be a KernelPlugin, in which case it is added straightaway and other parameters are ignored, a custom class that contains methods with the <code>kernel_function</code> decorator or a dictionary of functions with the <code>kernel_function</code> decorator for one or several methods. default value: None
plugin_name	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The name of the plugin, used if the plugin is not a KernelPlugin, if the plugin is None and the <code>parent_directory</code> is set, <code>KernelPlugin.from_directory</code> is called with those parameters, see <code>KernelPlugin.from_directory</code> for details. default value: None
parent_directory	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The parent directory path where the plugin directory resides default value: None
description	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The description of the plugin, used if the plugin is not a KernelPlugin.

Name	Description
	default value: None
class_init_arguments	<code>dict[str, dict[str, <xref: Any>]] <xref: None></code> The class initialization arguments default value: None

Returns

[Expand table](#)

Type	Description
<code>KernelPlugin</code>	The plugin that was added.

Exceptions

[Expand table](#)

Type	Description
<code>ValidationError</code>	If a KernelPlugin needs to be created, but it is not valid.

add_plugin_from_openai

Add a plugin from an OpenAPI document.

Python

```
async add_plugin_from_openai(plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None) -> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin_name Required*	<code>str</code> The name of the plugin
plugin_url	<code><xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>></code> The URL of the plugin default value: None
plugin_str	<code><xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>></code> The JSON string of the plugin default value: None
execution_parameters	<code><xref:<xref:semantic_kernel.functions.kernel_function_extension.OpenAIFunctionExecutionParameters</code>

Name	Description
	None>> The execution parameters default value: None
description	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The description of the plugin default value: None

Returns

[Expand table](#)

Type	Description
KernelPlugin	The imported plugin

Exceptions

[Expand table](#)

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

add_plugin_from_openapi

Add a plugin from the Open AI manifest.

Python

```
add_plugin_from_openapi(plugin_name: str, openapi_document_path: str, execution_settings: OpenAPIFunctionExecutionParameters | None = None, description: str | None = None) -> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin_name Required*	str The name of the plugin
plugin_url Required*	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The URL of the plugin
plugin_str Required*	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The JSON string of the plugin
execution_parameters Required*	<xref:<xref:semantic_kernel.functions.kernel_function_extension.OpenAIFunctionExecutionParameters

Name	Description
	None>> The execution parameters
openapi_document_path Required*	
execution_settings	default value: None
description	default value: None

Returns

[] Expand table

Type	Description
KernelPlugin	The imported plugin

Exceptions

[] Expand table

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

add_plugins

Adds a list of plugins to the kernel's collection of plugins.

Python

```
add_plugins(plugins: list[semantic_kernel.functions.kernel_plugin.KernelPlugin] | dict[str, semantic_kernel.functions.kernel_plugin.KernelPlugin | object]) -> None
```

Parameters

[] Expand table

Name	Description
plugins Required*	<code>list[<xref:KernelPlugin>]<xref: dict>[str,<xref: KernelPlugin>]</code> The plugins to add to the kernel

get_full_list_of_function_metadata

Get a list of all function metadata in the plugins.

Python

```
get_full_list_of_function_metadata() ->
list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMetadata]
```

get_function

Get a function by plugin_name and function_name.

Python

```
get_function(plugin_name: str | None, function_name: str) -> KernelFunction
```

Parameters

[+] Expand table

Name	Description
plugin_name Required*	<xref:<xref:semantic_kernel.functions.kernel_function_extension.str None>> The name of the plugin
function_name Required*	str The name of the function

Returns

[+] Expand table

Type	Description
KernelFunction	The function

Exceptions

[+] Expand table

Type	Description
KernelPluginNotFoundError	If the plugin is not found
KernelFunctionNotFoundError	If the function is not found

get_function_from_fully_qualified_function_name

Get a function by its fully qualified name (<plugin_name>-<function_name>).

Python

```
get_function_from_fully_qualified_function_name(fully_qualified_function_name: str) ->
KernelFunction
```

Parameters

[\[+\] Expand table](#)

Name	Description
<code>fully_qualified_function_name</code> Required*	<code>str</code> The fully qualified name of the function, if there is no '-' in the name, it is assumed that it is only a function_name.

Returns

[\[+\] Expand table](#)

Type	Description
<code>KernelFunction</code>	The function

Exceptions

[\[+\] Expand table](#)

Type	Description
<code>KernelPluginNotFoundError</code>	If the plugin is not found
<code>KernelFunctionNotFoundError</code>	If the function is not found

get_list_of_function_metadata

Get a list of all function metadata in the plugin collection.

Python

```
get_list_of_function_metadata(*args: Any, **kwargs: Any) ->
list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMetadata]
```

get_list_of_function_metadata_bool

Get a list of the function metadata in the plugin collection

Python

```
get_list_of_function_metadata_bool(include_prompt: bool = True, include_native: bool =
True) -> list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMetadata]
```

Parameters

[\[+\] Expand table](#)

Name	Description
<code>include_prompt</code>	<p><code>bool</code> Whether to include semantic functions in the list. default value: True</p>
<code>include_native</code>	<p><code>bool</code> Whether to include native functions in the list. default value: True</p>

Returns

[Expand table](#)

Type	Description
	A list of KernelFunctionMetadata objects in the collection.

get_list_of_function_metadata_filters

Get a list of Kernel Function Metadata based on filters.

Python

```
get_list_of_function_metadata_filters(filters: dict[Literal['excluded_plugins',
    'included_plugins', 'excluded_functions', 'included_functions'], list[str]]) ->
list[semantic_kernel.functions.kernel_function_metadata.KernelFunctionMetadata]
```

Parameters

[Expand table](#)

Name	Description
<code>filters</code> Required*	<p><code>dict[str,list[str]]</code> The filters to apply to the function list. The keys are:</p> <ul style="list-style-type: none"> • included_plugins: A list of plugin names to include. • excluded_plugins: A list of plugin names to exclude. • included_functions: A list of function names to include. • excluded_functions: A list of function names to exclude. <p>The included and excluded parameters are mutually exclusive. The function names are checked against the fully qualified name of a function.</p>

Returns

[Expand table](#)

Type	Description
<code>list[KernelFunctionMetadata]</code>	The list of Kernel Function Metadata that match the filters.

get_plugin

Get a plugin by name.

Python

```
get_plugin(plugin_name: str) -> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin_name Required*	str The name of the plugin

Returns

[Expand table](#)

Type	Description
KernelPlugin	The plugin

Exceptions

[Expand table](#)

Type	Description
KernelPluginNotFoundError	If the plugin is not found

rewrite_plugins

Rewrite plugins to a dictionary.

Python

```
rewrite_plugins(plugins: KernelPlugin |  
list[semantic_kernel.functions.kernel_plugin.KernelPlugin] | dict[str,  
semantic_kernel.functions.kernel_plugin.KernelPlugin] | None = None) -> dict[str,  
semantic_kernel.functions.kernel_plugin.KernelPlugin]
```

Parameters

[Expand table](#)

Name	Description
plugins	default value: None

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*]
[pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*]
[pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'plugins': FieldInfo(annotation=dict[str, KernelPlugin], required=False, default_factory=dict)}
```

plugins

Python

```
plugins: dict[str, semantic_kernel.functions.kernel_plugin.KernelPlugin]
```

kernel_function_from_method Module

Reference

Classes

 Expand table

KernelFunctionFromMethod	Semantic Kernel Function from a method. Initializes a new instance of the KernelFunctionFromMethod class
--	---

KernelFunctionFromMethod Class

Reference

Semantic Kernel Function from a method.

Initializes a new instance of the KernelFunctionFromMethod class

Inheritance [KernelFunction](#) → KernelFunctionFromMethod

Constructor

Python

```
KernelFunctionFromMethod(method: Callable[..., Any], plugin_name: str | None = None, stream_method: Callable[..., Any] | None = None, parameters: list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterMetadata] | None = None, return_parameter: KernelParameterMetadata | None = None, additional_metadata: dict[str, Any] | None = None)
```

Parameters

 [Expand table](#)

Name	Description
method Required*	<xref:Callable>[<xref:>, <xref: Any>] The method to be called
plugin_name	<xref:<xref:semantic_kernel.functions.kernel_function_from_method.str None>> The name of the plugin default value: None
stream_method	<xref:Callable>[<xref:>, <xref: Any>]<xref: None> The stream method for the function default value: None
parameters	list [<xref:KernelParameterMetadata>]<xref: None> The parameters of the function default value: None
return_parameter	<xref: <xref:semantic_kernel.functions.kernel_function_from_method.KernelParameterMetadata None>> The return parameter of the function default value: None
additional_metadata	dict [str,<xref: Any>]<xref: None> Additional metadata for the function

Name	Description
	default value: None

Methods

[Expand table](#)

gather_function_parameters	Gathers the function parameters from the arguments.
--	---

gather_function_parameters

Gathers the function parameters from the arguments.

Python

```
gather_function_parameters(context: FunctionInvocationContext) -> dict[str, Any]
```

Parameters

[Expand table](#)

Name	Description
context Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo] [pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':  
FieldInfo(annotation=KernelFunctionMetadata, required=True), 'method':  
FieldInfo(annotation=Callable[..., Any], required=True), 'stream_method':  
FieldInfo(annotation=Union[Callable[..., Any], NoneType], required=False,  
default=None)}
```

metadata

Python

```
metadata: KernelFunctionMetadata
```

method

Python

```
method: Callable[..., Any]
```

stream_method

Python

```
stream_method: Callable[..., Any] | None
```

kernel_function_from_prompt Module

Reference

Classes

 Expand table

KernelFunctionFromPrompt	Semantic Kernel Function from a prompt. Initializes a new instance of the KernelFunctionFromPrompt class
--	---

KernelFunctionFromPrompt Class

Reference

Semantic Kernel Function from a prompt.

Initializes a new instance of the KernelFunctionFromPrompt class

Inheritance [KernelFunction](#) → KernelFunctionFromPrompt

Constructor

Python

```
KernelFunctionFromPrompt(function_name: str, plugin_name: str | None = None,
description: str | None = None, prompt: str | None = None, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
prompt_template: PromptTemplateBase | None = None, prompt_template_config:
PromptTemplateConfig | None = None, prompt_execution_settings: None |
PromptExecutionSettings |
list[semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecution
Settings] | dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSetti
ngs] = None)
```

Parameters

[] [Expand table](#)

Name	Description
function_name Required*	str The name of the function
plugin_name	str The name of the plugin default value: None
description	str The description for the function default value: None
prompt	<xref:Optional>[str] The prompt default value: None

Name	Description
template_format	<xref:Optional>[str] The template format, default is "semantic-kernel" default value: semantic-kernel
prompt_template	<xref:Optional>[<xref:KernelPromptTemplate>] The prompt template default value: None
prompt_template_config	<xref:Optional>[<xref:PromptTemplateConfig>] The prompt template configuration default value: None
prompt_execution_settings	Optional instance, list or dict of PromptExecutionSettings to be used by the function, can also be supplied through prompt_template_config, but the supplied one is used if both are present. prompt_template_config (Optional[PromptTemplateConfig]): the prompt template config. default value: None

Methods

[\[\] Expand table](#)

from_directory	Creates a new instance of the KernelFunctionFromPrompt class from a directory. The directory needs to contain: <ul style="list-style-type: none">• A prompt file named <i>skprompt.txt</i>• A config file named <i>config.json</i>
from_yaml	Creates a new instance of the KernelFunctionFromPrompt class from a YAML string.
rewrite_execution_settings	Rewrite execution settings to a dictionary. If the prompt_execution_settings is not a dictionary, it is converted to a dictionary. If it is not supplied, but prompt_template is, the prompt_template's execution settings are used.
update_arguments_with_defaults	Update any missing values with their defaults.

from_directory

Creates a new instance of the KernelFunctionFromPrompt class from a directory.

The directory needs to contain:

- A prompt file named *skprompt.txt*
- A config file named *config.json*

Python

```
from_directory(path: str, plugin_name: str | None = None) ->
KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
path Required*	
plugin_name	default value: None

Returns

[\[\] Expand table](#)

Type	Description
KernelFunctionFromPrompt	The kernel function from prompt

from_yaml

Creates a new instance of the KernelFunctionFromPrompt class from a YAML string.

Python

```
from_yaml(yaml_str: str, plugin_name: str | None = None) ->
KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
yaml_str Required*	
plugin_name	default value: None

rewrite_execution_settings

Rewrite execution settings to a dictionary.

If the prompt_execution_settings is not a dictionary, it is converted to a dictionary. If it is not supplied, but prompt_template is, the prompt_template's execution settings are used.

Python

```
rewrite_execution_settings(data: dict[str, Any]) -> dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

Parameters

[\[\] Expand table](#)

Name	Description
data Required*	

update_arguments_with_defaults

Update any missing values with their defaults.

Python

```
update_arguments_with_defaults(arguments: KernelArguments) -> None
```

Parameters

Name	Description
arguments Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':
FieldInfo(annotation=KernelFunctionMetadata, required=True),
'prompt_execution_settings': FieldInfo(annotation=dict[str,
PromptExecutionSettings], required=False, default_factory=dict),
```

```
'prompt_template': FieldInfo(annotation=PromptTemplateBase,  
required=True)}
```

metadata

Python

```
metadata: KernelFunctionMetadata
```

prompt_execution_settings

Python

```
prompt_execution_settings: dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

prompt_template

Python

```
prompt_template: PromptTemplateBase
```

kernel_function_metadata Module

Reference

Classes

 [Expand table](#)

KernelFunctionMetadata	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
-------------------------------	--

KernelFunctionMetadata Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → KernelFunctionMetadata

Constructor

Python

```
KernelFunctionMetadata(*, name: str, plugin_name: str | None = None,  
description: str | None = None, parameters:  
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterMeta  
data] = None, is_prompt: bool, is_asynchronous: bool | None = True,  
return_parameter: KernelParameterMetadata | None = None,  
additional_properties: dict[str, Any] | None = None)
```

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
<code>name</code> Required*	
<code>plugin_name</code> Required*	
<code>description</code> Required*	
<code>parameters</code> Required*	
<code>is_prompt</code> Required*	

Name	Description
<code>is_asynchronous</code>	default value: True
<code>return_parameter</code> Required*	
<code>additional_properties</code> Required*	

Attributes

fully_qualified_name

Get the fully qualified name of the function.

Returns

[\[+\] Expand table](#)

Type	Description
	The fully qualified name of the function.

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'additional_properties': FieldInfo(annotation=Union[dict[str, Any], NoneType], required=False, default=None), 'description': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'is_asynchronous': FieldInfo(annotation=Union[bool, NoneType], required=False, default=True), 'is_prompt': FieldInfo(annotation=bool, required=True), 'name': FieldInfo(annotation=str, required=True, metadata=[_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'parameters': FieldInfo(annotation=list[KernelParameterMetadata], required=False, default_factory=list), 'plugin_name': FieldInfo(annotation=Union[str, NoneType], required=False, default=None, metadata=[_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'return_parameter': FieldInfo(annotation=Union[KernelParameterMetadata, NoneType], required=False, default=None)}
```

additional_properties

Python

```
additional_properties: dict[str, Any] | None
```

description

Python

```
description: str | None
```

is_asynchronous

Python

```
is_asynchronous: bool | None
```

is_prompt

Python

```
is_prompt: bool
```

name

Python

```
name: str
```

parameters

Python

```
parameters:  
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterM  
etadata]
```

plugin_name

Python

```
plugin_name: str | None
```

return_parameter

Python

```
return_parameter: KernelParameterMetadata | None
```

kernel_parameter_metadata Module

Reference

Classes

 Expand table

KernelParameterMetadata	<p>Create a new model by parsing and validating input data from keyword arguments.</p> <p>Raises <code>[ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><i>self</i> is explicitly positional-only to allow <i>self</i> as a field name.</p>
--------------------------------	--

KernelParameterMetadata Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → KernelParameterMetadata

Constructor

Python

```
KernelParameterMetadata(*, name: str | None = None, description: str | None = None, default_value: Any | None = None, type: str | None = 'str', is_required: bool | None = False, type_object: Any | None = None, schema_data: dict[str, Any] | None = None)
```

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
name Required*	
description Required*	
default_value Required*	
type	default value: str
is_required Required*	
type_object Required*	
schema_data	

Name	Description
Required*	

Methods

[\[\] Expand table](#)

form_schema
infer_schema

form_schema

Python

```
form_schema(data: Any) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
data Required*	

infer_schema

Python

```
infer_schema(type_object: type | None, parameter_type: str | None,  
default_value: Any, description: str | None) -> dict[str, Any] | None
```

Parameters

[\[\] Expand table](#)

Name	Description
type_object	

Name	Description
Required*	
parameter_type	
Required*	
default_value	
Required*	
description	
Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'default_value':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'description': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'is_required': FieldInfo(annotation=Union[bool, NoneType],  
    required=False, default=False), 'name': FieldInfo(annotation=Union[str,  
    NoneType], required=True, metadata=  
    [_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'schema_data':  
    FieldInfo(annotation=Union[dict[str, Any], NoneType], required=False,  
    default=None), 'type_': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default='str', alias='type', alias_priority=2),  
    'type_object': FieldInfo(annotation=Union[Any, NoneType], required=False,  
    default=None)}
```

default_value

Python

```
default_value: Any | None
```

description

Python

```
description: str | None
```

is_required

Python

```
is_required: bool | None
```

name

Python

```
name: str | None
```

schema_data

Python

```
schema_data: dict[str, Any] | None
```

type_

Python

```
type_: str | None
```

type_object

Python

```
type_object: Any | None
```

kernel_plugin Module

Reference

Classes

[\[\] Expand table](#)

[KernelPlugin](#) Represents a Kernel Plugin with functions.

This class behaves mostly like a dictionary, with functions as values and their names as keys. When you add a function, through `.set` or `setitem`, the function is copied, the metadata is deep-copied and the name of the plugin is set in the metadata and added to the dict of functions. This is done in the same way as a normal dict, so a existing key will be overwritten.

Class methods: `from_object(plugin_name: str, plugin_instance: Any | dict[str, Any], description: str | None = None)`: Create a plugin from a existing object, like a custom class with annotated functions.

`from_directory(plugin_name: str, parent_directory: str, description: str | None = None)`: Create a plugin from a directory, parsing: .py files, .yaml files and directories with skprompt.txt and config.json files.

`from.openapi(plugin_name: str, openapi_document_path: str, execution_settings: OpenAPIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from an OpenAPI document.

`from_openai(plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from the Open AI manifest.

Create a KernelPlugin

KernelPlugin Class

Reference

Represents a Kernel Plugin with functions.

This class behaves mostly like a dictionary, with functions as values and their names as keys. When you add a function, through `.set` or `setitem`, the function is copied, the metadata is deep-copied and the name of the plugin is set in the metadata and added to the dict of functions. This is done in the same way as a normal dict, so a existing key will be overwritten.

Class methods: `from_object(plugin_name: str, plugin_instance: Any | dict[str, Any], description: str | None = None)`: Create a plugin from a existing object, like a custom class with annotated functions.

`from_directory(plugin_name: str, parent_directory: str, description: str | None = None)`: Create a plugin from a directory, parsing: .py files, .yaml files and directories with skprompt.txt and config.json files.

`from_openapi(plugin_name: str, openapi_document_path: str, execution_settings: OpenAPIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from an OpenAPI document.

`from_openai(plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from the Open AI manifest.

Create a KernelPlugin

Inheritance [KernelBaseModel](#) → KernelPlugin

Constructor

Python

```
KernelPlugin(name: str, description: str | None = None, functions: Any]]] | None = None)
```

Parameters

[Expand table](#)

Name	Description
name Required*	The name of the plugin. The name can be upper/lower case letters and underscores.
description	The description of the plugin. default value: None
functions	The functions in the plugin, will be rewritten to a dictionary of functions. default value: None

Methods

[Expand table](#)

add	
add_dict	Add a dictionary of functions to the plugin.
add_list	Add a list of functions to the plugin.
from_directory	Create a plugin from a specified directory. This method does not recurse into subdirectories beyond one level deep from the specified plugin directory. For YAML files, function names are extracted from the content of the YAML files themselves (the name property). For directories, the function name is assumed to be the name of the directory. Each KernelFunction object is initialized with data parsed from the associated files and added to a list of functions that are then assigned to the created KernelPlugin object. A .py file is parsed and a plugin created, the functions within as then combined with any other functions found. The python file needs to contain a class with one or more kernel_function decorated methods. If this class has a <i>init</i> method, it will be called with the arguments provided in the <i>class_init_arguments</i> dictionary, the key needs to be the same as the name of the class, with the value being a dictionary of arguments to pass to the class (using kwargs). MyPlugins/ <<— pluginA.yaml <<— pluginB.yaml <<— native_function.py <<— Directory1/ >> <<— skprompt.txt >> <<— config.json <<— Directory2/ >> <<— skprompt.txt >> <<— config.json

Calling `KernelPlugin.from_directory("MyPlugins", "/path/to")` will create a `KernelPlugin` object named "MyPlugins", containing `KernelFunction` objects for `pluginA.yaml`, `pluginB.yaml`, `Directory1`, and `Directory2`, each initialized with their respective configurations. And functions for anything within `native_function.py`.

<code>from_object</code>	Creates a plugin that wraps the specified target object and imports it into the kernel's plugin collection
<code>from_openai</code>	Create a plugin from the Open AI manifest.
<code>from_openapi</code>	Create a plugin from an OpenAPI document.
<code>from_python_file</code>	
<code>get</code>	Get a function from the plugin.
<code>get_functions_metadata</code>	Get the metadata for the functions in the plugin.
<code>set</code>	Set a function in the plugin.
<code>setdefault</code>	Set a default value for a key.
<code>update</code>	Update the plugin with the functions from another.

add

Python

```
add(functions: Any) -> None
```

add_dict

Add a dictionary of functions to the plugin.

Python

```
add_dict(functions: dict[str,  
Union[semantic_kernel.functions.kernel_function.KernelFunction,  
collections.abc.Callable[..., Any]]]) -> None
```

Parameters

 Expand table

Name	Description
<code>functions</code> Required*	

add_list

Add a list of functions to the plugin.

Python

```
add_list(functions:  
list[Union[semantic_kernel.functions.kernel_function.KernelFunction,  
collections.abc.Callable[..., Any],  
semantic_kernel.functions.kernel_plugin.KernelPlugin]]) -> None
```

Parameters

 [Expand table](#)

Name	Description
functions Required*	

from_directory

Create a plugin from a specified directory.

This method does not recurse into subdirectories beyond one level deep from the specified plugin directory. For YAML files, function names are extracted from the content of the YAML files themselves (the name property). For directories, the function name is assumed to be the name of the directory. Each KernelFunction object is initialized with data parsed from the associated files and added to a list of functions that are then assigned to the created KernelPlugin object. A .py file is parsed and a plugin created, the functions within as then combined with any other functions found. The python file needs to contain a class with one or more kernel_function decorated methods. If this class has a *init* method, it will be called with the arguments provided in the *class_init_arguments* dictionary, the key needs to be the same as the name of the class, with the value being a dictionary of arguments to pass to the class (using kwargs).

MyPlugins/

|<<— pluginA.yaml |<<— pluginB.yaml |<<— native_function.py |<<— Directory1/

```
>>|<<— skprompt.txt  
>>|<<— config.json
```

|<<— Directory2/ >>|<<— skprompt.txt >>|<<— config.json

Calling `KernelPlugin.from_directory("MyPlugins", "/path/to")` will create a `KernelPlugin` object named "MyPlugins", containing `KernelFunction` objects for `pluginA.yaml`, `pluginB.yaml`, `Directory1`, and `Directory2`, each initialized with their respective configurations. And functions for anything within `native_function.py`.

Python

```
from_directory(plugin_name: str, parent_directory: str, description: str | None = None, class_init_arguments: dict[str, dict[str, Any]] | None = None) -> KernelPlugin
```

Parameters

[] [Expand table](#)

Name	Description
<code>plugin_name</code> Required*	<code>str</code> The name of the plugin, this is the name of the directory within the parent directory
<code>parent_directory</code> Required*	<code>str</code> The parent directory path where the plugin directory resides
<code>description</code>	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The description of the plugin default value: None
<code>class_init_arguments</code>	<code>dict[str,dict[str,<xref: Any>]] None</code> The class initialization arguments default value: None

Returns

[] [Expand table](#)

Type	Description
<code>KernelPlugin</code>	The created plugin of type <code>KernelPlugin</code> .

Exceptions

[] [Expand table](#)

Type	Description
<code>PluginInitializationError</code>	If the plugin directory does not exist.

Type	Description
PluginInvalidNameError	If the plugin name is invalid.

Examples

Assuming a plugin directory structure as follows:

from_object

Creates a plugin that wraps the specified target object and imports it into the kernel's plugin collection

Python

```
from_object(plugin_name: str, plugin_instance: Any | dict[str, Any], description: str | None = None) -> KernelPlugin
```

Parameters

[] [Expand table](#)

Name	Description
plugin_instance Required*	<xref:Any dict>[str,<xref: Any>] The plugin instance. This can be a custom class or a dictionary of classes that contains methods with the <code>kernel_function</code> decorator for one or several methods. See <code>TextMemoryPlugin</code> as an example.
plugin_name Required*	<code>str</code> The name of the plugin. Allows chars: upper, lower ASCII and underscores.
description	default value: None

Returns

[] [Expand table](#)

Type	Description
<code>KernelPlugin</code>	The imported plugin of type <code>KernelPlugin</code> .

from_openai

Create a plugin from the Open AI manifest.

Python

```
async classmethod from_openai(plugin_name: str, plugin_url: str | None = None,
plugin_str: str | None = None, execution_parameters:
OpenAIFunctionExecutionParameters | None = None, description: str | None = None)
-> KernelPlugin
```

Parameters

[] Expand table

Name	Description
plugin_name Required*	str The name of the plugin
plugin_url	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The URL of the plugin default value: None
plugin_str	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The JSON string of the plugin default value: None
execution_parameters	<xref:<xref:semantic_kernel.functions.kernel_plugin.OpenAIFunctionExecutionParameters None>> The execution parameters default value: None
description	default value: None

Returns

[] Expand table

Type	Description
KernelPlugin	The created plugin

Exceptions

[] Expand table

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

from_openapi

Create a plugin from an OpenAPI document.

Python

```
from_openapi(plugin_name: str, openapi_document_path: str, execution_settings:  
    OpenAIFunctionExecutionParameters | None = None, description: str | None = None)  
-> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin_name Required*	str The name of the plugin
plugin_url Required*	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The URL of the plugin
plugin_str Required*	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The JSON string of the plugin
execution_parameters Required*	<xref: <xref:semantic_kernel.functions.kernel_plugin.OpenAIFunctionExecutionParameters None>> The execution parameters
description	<xref:<xref:semantic_kernel.functions.kernel_plugin.str None>> The description of the plugin default value: None
openapi_document_path Required*	
execution_settings	default value: None

Returns

[Expand table](#)

Type	Description
KernelPlugin	The created plugin

Exceptions

[Expand table](#)

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

from_python_file

Python

```
from_python_file(plugin_name: str, py_file: str, description: str | None = None,  
class_init_arguments: dict[str, dict[str, Any]] | None = None) -> KernelPlugin
```

Parameters

[Expand table](#)

Name	Description
plugin_name Required*	
py_file Required*	
description	default value: None
class_init_arguments	default value: None

get

Get a function from the plugin.

Python

```
get()
```

Parameters

[Expand table](#)

Name	Description
key Required*	
default	default value: None

get_functions_metadata

Get the metadata for the functions in the plugin.

Python

```
get_functions_metadata()
```

set

Set a function in the plugin.

Python

```
set()
```

Parameters

[Expand table](#)

Name	Description
key Required*	
value Required*	

setdefault

Set a default value for a key.

Python

```
setdefault()
```

Parameters

[Expand table](#)

Name	Description
key Required*	

Name	Description
value	default value: None

update

Update the plugin with the functions from another.

Python

```
update()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*] [`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'description':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'functions': FieldInfo(annotation=dict[str, KernelFunction], required=False,
default_factory=dict), 'name': FieldInfo(annotation=str, required=True, metadata=}
```

```
[StringConstraints(strip_whitespace=None, to_upper=None, to_lower=None,  
strict=None, min_length=1, max_length=None, pattern='^[0-9A-Za-z_]+$')])}
```

name

The name of the plugin. The name can be upper/lower case letters and underscores.

Python

```
name: str
```

description

The description of the plugin.

Python

```
description: str | None
```

functions

The functions in the plugin, indexed by their name.

Python

```
functions: dict[str, semantic_kernel.functions.kernel_function.KernelFunction]
```

prompt_rendering_result Module

Reference

Classes

[+] Expand table

PromptRenderingResult Represents the result of rendering a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

PromptRenderingResult Class

Reference

Represents the result of rendering a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `PromptRenderingResult`

Constructor

Python

```
PromptRenderingResult(*, rendered_prompt: str, ai_service:  
    AIServiceClientBase, execution_settings: PromptExecutionSettings,  
    function_result: FunctionResult | None = None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>rendered_prompt</code> Required*	
<code>ai_service</code> Required*	
<code>execution_settings</code> Required*	
<code>function_result</code> Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_service':
FieldInfo(annotation=AIServiceClientBase, required=True),
'execution_settings': FieldInfo(annotation=PromptExecutionSettings,
required=True), 'function_result':
FieldInfo(annotation=Union[FunctionResult, NoneType], required=False,
default=None), 'rendered_prompt': FieldInfo(annotation=str,
required=True)}
```

rendered_prompt

The rendered prompt.

Python

```
rendered_prompt: str
```

ai_service

The AI service that rendered the prompt.

```
Python
```

```
ai_service: AIServiceClientBase
```

execution_settings

The execution settings for the prompt.

```
Python
```

```
execution_settings: PromptExecutionSettings
```

function_result

The result of executing the prompt.

```
Python
```

```
function_result: FunctionResult | None
```

types Module

Reference

FunctionResult Class

Reference

The result of a function.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → FunctionResult

Constructor

Python

```
FunctionResult(*, function: KernelFunctionMetadata, value: Any, metadata: dict[str, Any] = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
function Required*	KernelFunctionMetadata The metadata of the function that was invoked.
value Required*	Any The value of the result.
metadata Required*	<code><xref:Mapping>[str,<xref: Any>]</code> The metadata of the result.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
function Required*	
value Required*	
metadata Required*	

Methods

[] [Expand table](#)

get_inner_content	Get the inner content of the function result when that is a KernelContent or subclass of the first item of the value if it is a list.
-----------------------------------	---

get_inner_content

Get the inner content of the function result when that is a KernelContent or subclass of the first item of the value if it is a list.

Python

```
get_inner_content()
```

Parameters

[] [Expand table](#)

Name	Description
index	default value: 0

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'function':  
FieldInfo(annotation=KernelFunctionMetadata, required=True), 'metadata':  
FieldInfo(annotation=dict[str, Any], required=False,  
default_factory=dict), 'value': FieldInfo(annotation=Any, required=True)}
```

function

Python

```
function: KernelFunctionMetadata
```

metadata

Python

```
metadata: dict[str, Any]
```

value

Python

```
value: Any
```

KernelArguments Class

Reference

Initializes a new instance of the KernelArguments class, this is a dict-like class with the additional field for the execution_settings.

This class is derived from a dict, hence behaves the same way, just adds the execution_settings as a dict, with service_id and the settings.

Inheritance builtins.dict → KernelArguments

Constructor

Python

```
KernelArguments(settings: PromptExecutionSettings |  
list[PromptExecutionSettings] | dict[str, PromptExecutionSettings] | None =  
None, **kwargs: Any)
```

Parameters

[] Expand table

Name	Description
settings	<xref:PromptExecutionSettings List>[<xref:PromptExecutionSettings>]<xref: None> The settings for the execution. If a list is given, make sure all items in the list have a unique service_id as that is used as the key for the dict. default value: None
**kwargs Required*	dict[str,<xref: Any>]

KernelFunction Class

Reference

Semantic Kernel function.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `KernelFunction`

Constructor

Python

```
KernelFunction(*, metadata: KernelFunctionMetadata)
```

Keyword-Only Parameters

[] Expand table

Name	Description
<code>metadata</code> Required*	

Methods

[] Expand table

<code>from_method</code>	Create a new instance of the <code>KernelFunctionFromMethod</code> class.
<code>from_prompt</code>	Create a new instance of the <code>KernelFunctionFromPrompt</code> class.
<code>function_copy</code>	Copy the function, can also override the <code>plugin_name</code> .
<code>invoke</code>	Invoke the function with the given arguments.

`invoke_stream` Invoke a stream async function with the given arguments.

from_method

Create a new instance of the KernelFunctionFromMethod class.

Python

```
from_method(method: Callable[..., Any], plugin_name: str | None = None,
            stream_method: Callable[..., Any] | None = None) ->
KernelFunctionFromMethod
```

Parameters

[\[\] Expand table](#)

Name	Description
method Required*	
plugin_name	default value: None
stream_method	default value: None

from_prompt

Create a new instance of the KernelFunctionFromPrompt class.

Python

```
from_prompt(function_name: str, plugin_name: str, description: str | None
= None, prompt: str | None = None, template_format: Literal['semantic-
kernel', 'handlebars', 'jinja2'] = 'semantic-kernel', prompt_template:
PromptTemplateBase | None = None, prompt_template_config:
PromptTemplateConfig | None = None, prompt_execution_settings:
PromptExecutionSettings | list[PromptExecutionSettings] | dict[str,
PromptExecutionSettings] | None = None) -> KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
function_name Required*	
plugin_name Required*	
description	default value: None
prompt	default value: None
template_format	default value: semantic-kernel
prompt_template	default value: None
prompt_template_config	default value: None
prompt_execution_settings	default value: None

function_copy

Copy the function, can also override the plugin_name.

Python

```
function_copy(plugin_name: str | None = None) -> KernelFunction
```

Parameters

[\[+\]](#) Expand table

Name	Description
plugin_name	str The new plugin name. default value: None

Returns

[\[+\]](#) Expand table

Type	Description
KernelFunction	The copied function.

invoke

Invoke the function with the given arguments.

Python

```
async invoke(kernel: Kernel, arguments: KernelArguments | None = None,
metadata: dict[str, Any] = {}, **kwargs: Any) -> FunctionResult | None
```

Parameters

[\[+\] Expand table](#)

Name	Description
kernel Required*	<xref:semantic_kernel.functions.Kernel> The kernel
arguments	KernelArguments The Kernel arguments default value: None
kwargs Required*	Any Additional keyword arguments that will be added to the KernelArguments.
metadata	default value: {}

Returns

[\[+\] Expand table](#)

Type	Description
FunctionResult	The result of the function

invoke_stream

Invoke a stream async function with the given arguments.

Python

```
async invoke_stream(kernel: Kernel, arguments: KernelArguments | None = None,
metadata: dict[str, Any] = {}, **kwargs: Any) ->
AsyncGenerator[FunctionResult | list[StreamingContentMixin | Any], Any]
```

Parameters

[] Expand table

Name	Description
kernel Required*	<xref:semantic_kernel.functions.Kernel> The kernel
arguments	KernelArguments The Kernel arguments default value: None
kwargs Required*	Any Additional keyword arguments that will be added to the KernelArguments.
metadata	default value: {}

Attributes

description

The description of the function.

fully_qualified_name

is_prompt

Whether the function is semantic.

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':  
FieldInfo(annotation=KernelFunctionMetadata, required=True)}
```

name

The name of the function. Must be upper/lower case letters and underscores with a minimum length of 1.

parameters

The parameters for the function.

plugin_name

The name of the plugin that contains this function. Must be upper/lower case letters and underscores with a minimum length of 1.

return_parameter

The return parameter for the function.

stream_function

The stream function for the function.

function

The function to call.

prompt_execution_settings

The AI prompt execution settings.

prompt_template_config

The prompt template configuration.

metadata

The metadata for the function.

Python

```
metadata: KernelFunctionMetadata
```

KernelFunctionFromMethod Class

Reference

Semantic Kernel Function from a method.

Initializes a new instance of the KernelFunctionFromMethod class

Inheritance [KernelFunction](#) → KernelFunctionFromMethod

Constructor

Python

```
KernelFunctionFromMethod(method: Callable[..., Any], plugin_name: str | None = None, stream_method: Callable[..., Any] | None = None, parameters: list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterMetadata] | None = None, return_parameter: KernelParameterMetadata | None = None, additional_metadata: dict[str, Any] | None = None)
```

Parameters

[] [Expand table](#)

Name	Description
method Required*	<xref:Callable>[<xref:>, <xref: Any>] The method to be called
plugin_name	<xref:<xref:semantic_kernel.functions.str None>> The name of the plugin default value: None
stream_method	<xref:Callable>[<xref:>, <xref: Any>]<xref: None>> The stream method for the function default value: None
parameters	list[<xref:KernelParameterMetadata>] <xref: None>> The parameters of the function default value: None
return_parameter	<xref:<xref:semantic_kernel.functions.KernelParameterMetadata None>> The return parameter of the function default value: None

Name	Description
additional_metadata	<code>dict[str, <xref: Any>] <xref: None></code> Additional metadata for the function default value: None

Methods

[\[\] Expand table](#)

gather_function_parameters	Gathers the function parameters from the arguments.
--	---

gather_function_parameters

Gathers the function parameters from the arguments.

Python

```
gather_function_parameters(context: FunctionInvocationContext) ->
    dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
context Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':  
FieldInfo(annotation=KernelFunctionMetadata, required=True), 'method':  
FieldInfo(annotation=Callable[..., Any], required=True), 'stream_method':  
FieldInfo(annotation=Union[Callable[..., Any], NoneType], required=False,  
default=None)}
```

method

Python

```
method: Callable[..., Any]
```

stream_method

Python

```
stream_method: Callable[..., Any] | None
```

KernelFunctionFromPrompt Class

Reference

Semantic Kernel Function from a prompt.

Initializes a new instance of the KernelFunctionFromPrompt class

Inheritance [KernelFunction](#) → KernelFunctionFromPrompt

Constructor

Python

```
KernelFunctionFromPrompt(function_name: str, plugin_name: str | None = None,
description: str | None = None, prompt: str | None = None, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
prompt_template: PromptTemplateBase | None = None, prompt_template_config:
PromptTemplateConfig | None = None, prompt_execution_settings: None |
PromptExecutionSettings |
list[semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecution
Settings] | dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSetti
ngs] = None)
```

Parameters

[] [Expand table](#)

Name	Description
function_name Required*	str The name of the function
plugin_name	str The name of the plugin default value: None
description	str The description for the function default value: None
prompt	<xref:Optional>[str] The prompt default value: None

Name	Description
template_format	<xref:Optional>[str] The template format, default is "semantic-kernel" default value: semantic-kernel
prompt_template	<xref:Optional>[<xref:KernelPromptTemplate>] The prompt template default value: None
prompt_template_config	<xref:Optional>[<xref:PromptTemplateConfig>] The prompt template configuration default value: None
prompt_execution_settings	Optional instance, list or dict of PromptExecutionSettings to be used by the function, can also be supplied through prompt_template_config, but the supplied one is used if both are present. prompt_template_config (Optional[PromptTemplateConfig]): the prompt template config. default value: None

Methods

[\[\] Expand table](#)

from_directory	Creates a new instance of the KernelFunctionFromPrompt class from a directory. The directory needs to contain: <ul style="list-style-type: none">• A prompt file named <i>skprompt.txt</i>• A config file named <i>config.json</i>
from_yaml	Creates a new instance of the KernelFunctionFromPrompt class from a YAML string.
rewrite_execution_settings	Rewrite execution settings to a dictionary. If the prompt_execution_settings is not a dictionary, it is converted to a dictionary. If it is not supplied, but prompt_template is, the prompt_template's execution settings are used.
update_arguments_with_defaults	Update any missing values with their defaults.

from_directory

Creates a new instance of the KernelFunctionFromPrompt class from a directory.

The directory needs to contain:

- A prompt file named *skprompt.txt*
- A config file named *config.json*

Python

```
from_directory(path: str, plugin_name: str | None = None) ->
KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
path Required*	
plugin_name	default value: None

Returns

[\[\] Expand table](#)

Type	Description
KernelFunctionFromPrompt	The kernel function from prompt

from_yaml

Creates a new instance of the KernelFunctionFromPrompt class from a YAML string.

Python

```
from_yaml(yaml_str: str, plugin_name: str | None = None) ->
KernelFunctionFromPrompt
```

Parameters

[\[\] Expand table](#)

Name	Description
yaml_str Required*	
plugin_name	default value: None

rewrite_execution_settings

Rewrite execution settings to a dictionary.

If the prompt_execution_settings is not a dictionary, it is converted to a dictionary. If it is not supplied, but prompt_template is, the prompt_template's execution settings are used.

Python

```
rewrite_execution_settings(data: dict[str, Any]) -> dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

Parameters

[\[\] Expand table](#)

Name	Description
data Required*	

update_arguments_with_defaults

Update any missing values with their defaults.

Python

```
update_arguments_with_defaults(arguments: KernelArguments) -> None
```

Parameters

Name	Description
arguments Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'metadata':
FieldInfo(annotation=KernelFunctionMetadata, required=True),
'prompt_execution_settings': FieldInfo(annotation=dict[str,
PromptExecutionSettings], required=False, default_factory=dict),
```

```
'prompt_template': FieldInfo(annotation=PromptTemplateBase,  
required=True)}
```

prompt_execution_settings

Python

```
prompt_execution_settings: dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

prompt_template

Python

```
prompt_template: PromptTemplateBase
```

KernelFunctionMetadata Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → KernelFunctionMetadata

Constructor

Python

```
KernelFunctionMetadata(*, name: str, plugin_name: str | None = None,  
description: str | None = None, parameters:  
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterMeta  
data] = None, is_prompt: bool, is_asynchronous: bool | None = True,  
return_parameter: KernelParameterMetadata | None = None,  
additional_properties: dict[str, Any] | None = None)
```

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
<code>name</code> Required*	
<code>plugin_name</code> Required*	
<code>description</code> Required*	
<code>parameters</code> Required*	
<code>is_prompt</code> Required*	

Name	Description
<code>is_asynchronous</code>	default value: True
<code>return_parameter</code> Required*	
<code>additional_properties</code> Required*	

Attributes

fully_qualified_name

Get the fully qualified name of the function.

Returns

[\[+\] Expand table](#)

Type	Description
	The fully qualified name of the function.

model_computed_fields

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [`ConfigDict`][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'additional_properties': FieldInfo(annotation=Union[dict[str, Any], NoneType], required=False, default=None), 'description': FieldInfo(annotation=Union[str, NoneType], required=False, default=None), 'is_asynchronous': FieldInfo(annotation=Union[bool, NoneType], required=False, default=True), 'is_prompt': FieldInfo(annotation=bool, required=True), 'name': FieldInfo(annotation=str, required=True, metadata=[_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'parameters': FieldInfo(annotation=list[KernelParameterMetadata], required=False, default_factory=list), 'plugin_name': FieldInfo(annotation=Union[str, NoneType], required=False, default=None, metadata=[_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'return_parameter': FieldInfo(annotation=Union[KernelParameterMetadata, NoneType], required=False, default=None)}
```

additional_properties

Python

```
additional_properties: dict[str, Any] | None
```

description

Python

```
description: str | None
```

is_asynchronous

Python

```
is_asynchronous: bool | None
```

is_prompt

Python

```
is_prompt: bool
```

name

Python

```
name: str
```

parameters

Python

```
parameters:  
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterM  
etadata]
```

plugin_name

Python

```
plugin_name: str | None
```

return_parameter

Python

```
return_parameter: KernelParameterMetadata | None
```

KernelParameterMetadata Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → KernelParameterMetadata

Constructor

Python

```
KernelParameterMetadata(*, name: str | None = None, description: str | None = None, default_value: Any | None = None, type: str | None = 'str', is_required: bool | None = False, type_object: Any | None = None, schema_data: dict[str, Any] | None = None)
```

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
name Required*	
description Required*	
default_value Required*	
type	default value: str
is_required Required*	
type_object Required*	
schema_data	

Name	Description
Required*	

Methods

[\[\] Expand table](#)

form_schema
infer_schema

form_schema

Python

```
form_schema(data: Any) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
data Required*	

infer_schema

Python

```
infer_schema(type_object: type | None, parameter_type: str | None,  
default_value: Any, description: str | None) -> dict[str, Any] | None
```

Parameters

[\[\] Expand table](#)

Name	Description
type_object	

Name	Description
Required*	
parameter_type	
Required*	
default_value	
Required*	
description	
Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'default_value':  
    FieldInfo(annotation=Union[Any, NoneType], required=False, default=None),  
    'description': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default=None), 'is_required': FieldInfo(annotation=Union[bool, NoneType],  
    required=False, default=False), 'name': FieldInfo(annotation=Union[str,  
    NoneType], required=True, metadata=  
    [_PydanticGeneralMetadata(pattern='^[0-9A-Za-z_]+$')]), 'schema_data':  
    FieldInfo(annotation=Union[dict[str, Any], NoneType], required=False,  
    default=None), 'type_': FieldInfo(annotation=Union[str, NoneType],  
    required=False, default='str', alias='type', alias_priority=2),  
    'type_object': FieldInfo(annotation=Union[Any, NoneType], required=False,  
    default=None)}
```

default_value

Python

```
default_value: Any | None
```

description

Python

```
description: str | None
```

is_required

Python

```
is_required: bool | None
```

name

Python

```
name: str | None
```

schema_data

Python

```
schema_data: dict[str, Any] | None
```

type_

Python

```
type_: str | None
```

type_object

Python

```
type_object: Any | None
```

KernelPlugin Class

Reference

Represents a Kernel Plugin with functions.

This class behaves mostly like a dictionary, with functions as values and their names as keys. When you add a function, through `.set` or `setitem`, the function is copied, the metadata is deep-copied and the name of the plugin is set in the metadata and added to the dict of functions. This is done in the same way as a normal dict, so a existing key will be overwritten.

Class methods: `from_object(plugin_name: str, plugin_instance: Any | dict[str, Any], description: str | None = None)`: Create a plugin from a existing object, like a custom class with annotated functions.

`from_directory(plugin_name: str, parent_directory: str, description: str | None = None)`: Create a plugin from a directory, parsing: .py files, .yaml files and directories with skprompt.txt and config.json files.

`from_openapi(plugin_name: str, openapi_document_path: str, execution_settings: OpenAPIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from an OpenAPI document.

`from_openai(plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None)`:

Create a plugin from the Open AI manifest.

Create a KernelPlugin

Inheritance [KernelBaseModel](#) → KernelPlugin

Constructor

Python

```
KernelPlugin(name: str, description: str | None = None, functions: Any]]] | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
name Required*	The name of the plugin. The name can be upper/lower case letters and underscores.
description	The description of the plugin. default value: None
functions	The functions in the plugin, will be rewritten to a dictionary of functions. default value: None

Methods

[\[\] Expand table](#)

add	
add_dict	Add a dictionary of functions to the plugin.
add_list	Add a list of functions to the plugin.
from_directory	Create a plugin from a specified directory. This method does not recurse into subdirectories beyond one level deep from the specified plugin directory. For YAML files, function names are extracted from the content of the YAML files themselves (the name property). For directories, the function name is assumed to be the name of the directory. Each KernelFunction object is initialized with data parsed from the associated files and added to a list of functions that are then assigned to the created KernelPlugin object. A .py file is parsed and a plugin created, the functions within as then combined with any other functions found. The python file needs to contain a class with one or more kernel_function decorated methods. If this class has a <i>init</i> method, it will be called with the arguments provided in the <i>class_init_arguments</i> dictionary, the key needs to be the same as the name of the class, with the value being a dictionary of arguments to pass to the class (using kwargs).

MyPlugins/

```
|<<— pluginA.yaml |<<— pluginB.yaml |<<—  
native_function.py |<<— Directory1/
```

```
>>|<<— skprompt.txt  
>>|<<— config.json
```

```
|<<— Directory2/ >>|<<— skprompt.txt >>|<<— config.json
```

Calling `KernelPlugin.from_directory("MyPlugins", "/path/to")` will create a `KernelPlugin` object named "MyPlugins", containing `KernelFunction` objects for `pluginA.yaml`, `pluginB.yaml`, `Directory1`, and `Directory2`, each initialized with their respective configurations. And functions for anything within `native_function.py`.

<code>from_object</code>	Creates a plugin that wraps the specified target object and imports it into the kernel's plugin collection
<code>from_openai</code>	Create a plugin from the Open AI manifest.
<code>from_openapi</code>	Create a plugin from an OpenAPI document.
<code>from_python_file</code>	
<code>get</code>	Get a function from the plugin.
<code>get_functions_metadata</code>	Get the metadata for the functions in the plugin.
<code>set</code>	Set a function in the plugin.
<code>setdefault</code>	Set a default value for a key.
<code>update</code>	Update the plugin with the functions from another.

add

```
Python
```

```
add(functions: Any) -> None
```

add_dict

Add a dictionary of functions to the plugin.

```
Python
```

```
add_dict(functions: dict[str,  
Union[semantic_kernel.functions.kernel_function.KernelFunction,  
collections.abc.Callable[..., Any]]]) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
functions Required*	

add_list

Add a list of functions to the plugin.

Python

```
add_list(functions:  
list[Union[semantic_kernel.functions.kernel_function.KernelFunction,  
collections.abc.Callable[..., Any],  
semantic_kernel.functions.kernel_plugin.KernelPlugin]]) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
functions Required*	

from_directory

Create a plugin from a specified directory.

This method does not recurse into subdirectories beyond one level deep from the specified plugin directory. For YAML files, function names are extracted from the content of the YAML files themselves (the name property). For directories, the function name is assumed to be the name of the directory. Each KernelFunction

object is initialized with data parsed from the associated files and added to a list of functions that are then assigned to the created KernelPlugin object. A .py file is parsed and a plugin created, the functions within as then combined with any other functions found. The python file needs to contain a class with one or more kernel_function decorated methods. If this class has a *init* method, it will be called with the arguments provided in the *class_init_arguments* dictionary, the key needs to be the same as the name of the class, with the value being a dictionary of arguments to pass to the class (using kwargs).

MyPlugins/

```
|<<— pluginA.yaml |<<— pluginB.yaml |<<— native_function.py |<<—  
| Directory1/
```

```
>>|<<— skprompt.txt  
>>|<<— config.json
```

```
|<<— Directory2/ >>|<<— skprompt.txt >>|<<— config.json
```

Calling *KernelPlugin.from_directory*("MyPlugins", "/path/to") will create a KernelPlugin object named "MyPlugins", containing KernelFunction objects for *pluginA.yaml*, *pluginB.yaml*, *Directory1*, and *Directory2*, each initialized with their respective configurations. And functions for anything within *native_function.py*.

Python

```
from_directory(plugin_name: str, parent_directory: str, description: str  
| None = None, class_init_arguments: dict[str, dict[str, Any]] | None =  
None) -> KernelPlugin
```

Parameters

[] Expand table

Name	Description
plugin_name Required*	<code>str</code> The name of the plugin, this is the name of the directory within the parent directory
parent_directory	<code>str</code>

Name	Description
Required*	The parent directory path where the plugin directory resides
description	<xref:<xref:semantic_kernel.functions.str None>> The description of the plugin default value: None
class_init_arguments	<code>dict[str, dict[str, <xref: Any>]] <xref: None></code> The class initialization arguments default value: None

Returns

[\[\] Expand table](#)

Type	Description
<code>KernelPlugin</code>	The created plugin of type <code>KernelPlugin</code> .

Exceptions

[\[\] Expand table](#)

Type	Description
<code>PluginInitializationError</code>	If the plugin directory does not exist.
<code>PluginInvalidNameError</code>	If the plugin name is invalid.

Examples

Assuming a plugin directory structure as follows:

from_object

Creates a plugin that wraps the specified target object and imports it into the kernel's plugin collection

Python

```
from_object(plugin_name: str, plugin_instance: Any | dict[str, Any],
description: str | None = None) -> KernelPlugin
```

Parameters

[+] Expand table

Name	Description
plugin_instance Required*	<xref:Any dict>[str ,<xref: Any>] The plugin instance. This can be a custom class or a dictionary of classes that contains methods with the <code>kernel_function</code> decorator for one or several methods. See <code>TextMemoryPlugin</code> as an example.
plugin_name Required*	str The name of the plugin. Allows chars: upper, lower ASCII and underscores.
description	default value: None

Returns

[+] Expand table

Type	Description
KernelPlugin	The imported plugin of type <code>KernelPlugin</code> .

from_openai

Create a plugin from the Open AI manifest.

Python

```
async classmethod from_openai(plugin_name: str, plugin_url: str | None = None, plugin_str: str | None = None, execution_parameters: OpenAIFunctionExecutionParameters | None = None, description: str | None = None) -> KernelPlugin
```

Parameters

[+] Expand table

Name	Description
plugin_name Required*	str The name of the plugin

Name	Description
plugin_url	<xref:<xref:semantic_kernel.functions.str None>> The URL of the plugin default value: None
plugin_str	<xref:<xref:semantic_kernel.functions.str None>> The JSON string of the plugin default value: None
execution_parameters	<xref:<xref:semantic_kernel.functions.OpenAIFunctionExecutionParameters None>> The execution parameters default value: None
description	default value: None

Returns

[] Expand table

Type	Description
KernelPlugin	The created plugin

Exceptions

[] Expand table

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

from_openapi

Create a plugin from an OpenAPI document.

Python

```
from_openapi(plugin_name: str, openapi_document_path: str,
execution_settings: OpenAIFunctionExecutionParameters | None = None,
description: str | None = None) -> KernelPlugin
```

Parameters

[\[\] Expand table](#)

Name	Description
plugin_name Required*	str The name of the plugin
plugin_url Required*	<xref:<xref:semantic_kernel.functions.str None>> The URL of the plugin
plugin_str Required*	<xref:<xref:semantic_kernel.functions.str None>> The JSON string of the plugin
execution_parameters Required*	<xref:<xref:semantic_kernel.functions.OpenAIFunctionExecutionParameters None>> The execution parameters
description	<xref:<xref:semantic_kernel.functions.str None>> The description of the plugin default value: None
openapi_document_path Required*	
execution_settings	default value: None

Returns

[\[\] Expand table](#)

Type	Description
KernelPlugin	The created plugin

Exceptions

[\[\] Expand table](#)

Type	Description
PluginInitializationError	if the plugin URL or plugin JSON/YAML is not provided

from_python_file

Python

```
from_python_file(plugin_name: str, py_file: str, description: str | None = None, class_init_arguments: dict[str, dict[str, Any]] | None = None) -> KernelPlugin
```

Parameters

[\[\] Expand table](#)

Name	Description
plugin_name Required*	
py_file Required*	
description	default value: None
class_init_arguments	default value: None

get

Get a function from the plugin.

Python

```
get()
```

Parameters

[\[\] Expand table](#)

Name	Description
key Required*	
default	default value: None

get_functions_metadata

Get the metadata for the functions in the plugin.

Python

```
get_functions_metadata()
```

set

Set a function in the plugin.

Python

```
set()
```

Parameters

[\[\] Expand table](#)

Name	Description
key Required*	
value Required*	

setdefault

Set a default value for a key.

Python

```
setdefault()
```

Parameters

[\[\] Expand table](#)

Name	Description
key Required*	
value	default value: None

update

Update the plugin with the functions from another.

Python

```
update()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'description':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=None),  
    'functions': FieldInfo(annotation=dict[str, KernelFunction],  
        required=False, default_factory=dict), 'name': FieldInfo(annotation=str,  
        required=True, metadata=[StringConstraints(strip_whitespace=None,  
            to_upper=None, to_lower=None, strict=None, min_length=1, max_length=None,  
            pattern='^[\u0-9A-Za-z_]+$')])}
```

name

The name of the plugin. The name can be upper/lower case letters and underscores.

Python

```
name: str
```

description

The description of the plugin.

Python

```
description: str | None
```

functions

The functions in the plugin, indexed by their name.

Python

```
functions: dict[str,  
semantic_kernel.functions.kernel_function.KernelFunction]
```

prompt_template Package

Reference

Packages

[\[\] Expand table](#)

utils

Modules

[\[\] Expand table](#)

const

handlebars_prompt_template
--

input_variable

jinja2_prompt_template
--

kernel_prompt_template
--

prompt_template_base

prompt_template_config
--

Classes

[\[\] Expand table](#)

HandlebarsPromptTemplate	Create a Handlebars prompt template.
--	--------------------------------------

Handlebars are parsed as a whole and therefore do not have variables that can be extracted, also with handlebars there is no distinction in syntax between a variable and a value, a value that is encountered is tried to resolve with the arguments and the functions, if not found, the literal value is returned.

Create a new model by parsing and validating input data from keyword arguments.

	<p><code>Raises [ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
InputVariable	<p>Input variable for a prompt template.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><code>Raises [ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
Jinja2PromptTemplate	<p>Creates and renders Jinja2 prompt templates to text.</p> <p>Jinja2 templates support advanced features such as variable substitution, control structures, and inheritance, making it possible to dynamically generate text based on input arguments and predefined functions. This class leverages Jinja2's flexibility to render prompts that can include conditional logic, loops, and functions, based on the provided template configuration and arguments.</p> <p>Note that the fully qualified function name (in the form of "plugin-function") is not allowed in Jinja2 because of the hyphen. Therefore, the function name is replaced with an underscore, which are allowed in Python function names.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><code>Raises [ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
KernelPromptTemplate	<p>Create a Kernel prompt template.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p> <p><code>Raises [ValidationError][pydantic_core.ValidationError]</code> if the input data cannot be validated to form a valid model.</p> <p><code>self</code> is explicitly positional-only to allow <code>self</code> as a field name.</p>
PromptTemplateConfig	<p>Configuration for a prompt template.</p> <p>Create a new model by parsing and validating input data from keyword arguments.</p>

`Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.`

`self` is explicitly positional-only to allow `self` as a field name.

utils Package

Reference

Modules

[+] Expand table

[handlebars_system_helpers](#)

[jinja2_system_helpers](#)

[template_function_helpers](#)

Functions

`create_template_helper_from_function`

Create a helper function for both the Handlebars and Jinja2 templating engines from a kernel function.

Python

```
create_template_helper_from_function(function: KernelFunction, kernel: Kernel, base_arguments: KernelArguments, template_format: Literal['handlebars', 'jinja2'], allow_dangerously_set_content: bool = False) -> Callable[..., Any]
```

Parameters

[+] Expand table

Name	Description
function Required*	
kernel Required*	
base_arguments Required*	

Name	Description
template_format Required*	
allow_dangerously_set_content	default value: False

handlebars_system_helpers Module

Reference

jinja2_system_helpers Module

Reference

template_function_helpers Module

Reference

Functions

create_template_helper_from_function

Create a helper function for both the Handlebars and Jinja2 templating engines from a kernel function.

Python

```
create_template_helper_from_function(function: KernelFunction, kernel: Kernel, base_arguments: KernelArguments, template_format: Literal['handlebars', 'jinja2'], allow_dangerously_set_content: bool = False) -> Callable[..., Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
function Required*	
kernel Required*	
base_arguments Required*	
template_format Required*	
allow_dangerously_set_content	default value: False

const Module

Reference

handlebars_prompt_template Module

Reference

Classes

[+] Expand table

[HandlebarsPromptTemplate](#) Create a Handlebars prompt template.

Handlebars are parsed as a whole and therefore do not have variables that can be extracted, also with handlebars there is no distinction in syntax between a variable and a value, a value that is encountered is tried to resolve with the arguments and the functions, if not found, the literal value is returned.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

HandlebarsPromptTemplate Class

Reference

Create a Handlebars prompt template.

Handlebars are parsed as a whole and therefore do not have variables that can be extracted, also with handlebars there is no distinction in syntax between a variable and a value, a value that is encountered is tried to resolve with the arguments and the functions, if not found, the literal value is returned.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [PromptTemplateBase](#) → HandlebarsPromptTemplate

Constructor

Python

```
HandlebarsPromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

[Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	<xref:semantic_kernel.prompt_template.handlebars_prompt_template.PromptTemplateConfig> The prompt template configuration This is checked if the template format is 'handlebars'
<code>allow_dangerously_set_content</code> Required*	<xref:<xref:semantic_kernel.prompt_template.handlebars_prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	
<code>allow_dangerously_set_content</code> Required*	

Methods

[Expand table](#)

model_post_init	We need to both initialize private attributes and call the user-defined model_post_init method.
render	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
validate_template_format	

model_post_init

We need to both initialize private attributes and call the user-defined model_post_init method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[Expand table](#)

Name	Description
_ModelMetaclass__context Required*	

render

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[Expand table](#)

Name	Description
kernel Required*	The kernel instance
arguments	The kernel arguments default value: None

Returns

[Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'allow_dangerously_set_content': FieldInfo(annotation=bool, required=False, default=False), 'prompt_template_config': FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

prompt_template_config

Python

```
prompt_template_config: PromptTemplateConfig
```

input_variable Module

Reference

Classes

 [Expand table](#)

InputVariable Input variable for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

InputVariable Class

Reference

Input variable for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `InputVariable`

Constructor

Python

```
InputVariable(*, name: str, description: str | None = '', default: Any | None = None, is_required: bool | None = True, json_schema: str | None = None, allow_dangerously_set_content: bool = False)
```

Parameters

[+] [Expand table](#)

Name	Description
<code>name</code> Required*	The name of the input variable.
<code>description</code> Required*	The description of the input variable.
<code>default</code> Required*	The default value of the input variable.
<code>is_required</code> Required*	Whether the input variable is required.
<code>json_schema</code> Required*	The JSON schema for the input variable.

Name	Description
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
name Required*	
description Required*	
default Required*	
is_required	default value: True
json_schema Required*	
allow_dangerously_set_content Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
'allow_dangerously_set_content': FieldInfo(annotation=bool,  
required=False, default=False), 'default':  
FieldInfo(annotation=Union[Any, NoneType], required=False, default=''),  
'description': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=''), 'is_required': FieldInfo(annotation=Union[bool, NoneType],  
required=False, default=True), 'json_schema':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=''),  
'name': FieldInfo(annotation=str, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

default

Python

```
default: Any | None
```

description

Python

```
description: str | None
```

is_required

Python

```
is_required: bool | None
```

json_schema

Python

```
json_schema: str | None
```

name

Python

```
name: str
```

jinja2_prompt_template Module

Reference

Classes

[] [Expand table](#)

[Jinja2PromptTemplate](#) Creates and renders Jinja2 prompt templates to text.

Jinja2 templates support advanced features such as variable substitution, control structures, and inheritance, making it possible to dynamically generate text based on input arguments and predefined functions. This class leverages Jinja2's flexibility to render prompts that can include conditional logic, loops, and functions, based on the provided template configuration and arguments.

Note that the fully qualified function name (in the form of "plugin-function") is not allowed in Jinja2 because of the hyphen. Therefore, the function name is replaced with an underscore, which are allowed in Python function names.

Create a new model by parsing and validating input data from keyword arguments.

Raises [[ValidationError](#)][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Jinja2PromptTemplate Class

Reference

Creates and renders Jinja2 prompt templates to text.

Jinja2 templates support advanced features such as variable substitution, control structures, and inheritance, making it possible to dynamically generate text based on input arguments and predefined functions. This class leverages Jinja2's flexibility to render prompts that can include conditional logic, loops, and functions, based on the provided template configuration and arguments.

Note that the fully qualified function name (in the form of "plugin-function") is not allowed in Jinja2 because of the hyphen. Therefore, the function name is replaced with an underscore, which are allowed in Python function names.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [PromptTemplateBase](#) → `Jinja2PromptTemplate`

Constructor

Python

```
Jinja2PromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

[Expand table](#)

Name	Description
prompt_template_config Required*	<xref:semantic_kernel.prompt_template.jinja2_prompt_template.PromptTemplateConfig> The configuration object for the prompt template. This should specify the template format as 'jinja2' and include any necessary configuration details required for rendering the template.
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.jinja2_prompt_template.bool = False>>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	
<code>allow_dangerously_set_content</code> Required*	

Methods

[Expand table](#)

<code>model_post_init</code>	We need to both initialize private attributes and call the user-defined <code>model_post_init</code> method.
<code>render</code>	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
<code>validate_template_format</code>	

`model_post_init`

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[Expand table](#)

Name	Description
<code>_ModelMetaclass__context</code> Required*	

`render`

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[Expand table](#)

Name	Description
kernel Required*	The kernel instance
arguments	The kernel arguments default value: None

Returns

[Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*]
[*pydantic.config.ConfigDict*].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo] [pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'allow_dangerously_set_content': FieldInfo(annotation=bool, required=False, default=False), 'prompt_template_config': FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

prompt_template_config

Python

```
prompt_template_config: PromptTemplateConfig
```

kernel_prompt_template Module

Reference

Classes

 [Expand table](#)

[KernelPromptTemplate](#) Create a Kernel prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

KernelPromptTemplate Class

Reference

Create a Kernel prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [PromptTemplateBase](#) → KernelPromptTemplate

Constructor

Python

```
KernelPromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

[\[+\] Expand table](#)

Name	Description
prompt_template_config Required*	<xref:semantic_kernel.prompt_template.kernel_prompt_template.PromptTemplateConfig> The prompt template configuration This includes the actual template to use.
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.kernel_prompt_template.bool = False>>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
prompt_template_config Required*	
allow_dangerously_set_content Required*	

Methods

[\[+\] Expand table](#)

extract_blocks	Given a prompt template string, extract all the blocks (text, variables, function calls).
model_post_init	We need to both initialize private attributes and call the user-defined model_post_init method.
render	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
render_blocks	Given a list of blocks render each block and compose the final result.
validate_template_format	

extract_blocks

Given a prompt template string, extract all the blocks (text, variables, function calls).

Python

```
extract_blocks() -> list[semantic_kernel.template_engine.blocks.block.Block]
```

Parameters

[\[+\] Expand table](#)

Name	Description
template_text Required*	Prompt template

Returns

[\[+\] Expand table](#)

Type	Description
	A list of all the blocks, ie the template tokenized in text, variables and function calls

model_post_init

We need to both initialize private attributes and call the user-defined model_post_init method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[\[+\] Expand table](#)

Name	Description
<code>_ModelMetaclass__context</code> Required*	

render

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[\[+\]](#) Expand table

Name	Description
<code>kernel</code> Required*	The kernel instance
<code>arguments</code>	The kernel arguments default value: None

Returns

[\[+\]](#) Expand table

Type	Description
	The prompt template ready to be used for an AI request

render_blocks

Given a list of blocks render each block and compose the final result.

Python

```
async render_blocks(blocks: list[semantic_kernel.template_engine.blocks.block.Block],  
kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

[\[+\]](#) Expand table

Name	Description
<code>blocks</code>	Template blocks generated by ExtractBlocks

Name	Description
Required*	
context Required*	Access into the current kernel execution context
kernel Required*	
arguments Required*	

Returns

[\[\] Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[\[\] Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True, 'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo] [pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'allow_dangerously_set_content': FieldInfo(annotation=bool, required=False, default=False), 'prompt_template_config': FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

prompt_template_config

Python

```
prompt_template_config: PromptTemplateConfig
```

prompt_template_base Module

Reference

Classes

[+] Expand table

<p><code>PromptTemplateBase</code></p>	<p>Create a new model by parsing and validating input data from keyword arguments.</p>
--	--

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

PromptTemplateBase Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [KernelBaseModel](#) → PromptTemplateBase

[ABC](#) → PromptTemplateBase

Constructor

Python

```
PromptTemplateBase(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	
<code>allow_dangerously_set_content</code> Required*	

Methods

[\[\] Expand table](#)

[render](#)

render

Python

```
abstract async render(kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	
arguments Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*] [`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'allow_dangerously_set_content': FieldInfo(annotation=bool,  
required=False, default=False), 'prompt_template_config':  
FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

prompt_template_config

Python

```
prompt_template_config: PromptTemplateConfig
```

prompt_template_config Module

Reference

Classes

 [Expand table](#)

[PromptTemplateConfig](#) Configuration for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

PromptTemplateConfig Class

Reference

Configuration for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `PromptTemplateConfig`

Constructor

Python

```
PromptTemplateConfig(*, name: str = '', description: str | None = '',
template: str | None = None, template_format: Literal['semantic-kernel',
'handlebars', 'jinja2'] = 'semantic-kernel', input_variables:
list[semantic_kernel.prompt_template.input_variable.InputVariable] = None,
allow_dangerously_set_content: bool = False, execution_settings: dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSetti
ngs] = None)
```

Parameters

[] Expand table

Name	Description
name Required*	The name of the prompt template.
description Required*	The description of the prompt template.
template Required*	The template for the prompt.
template_format Required*	The format of the template, should be 'semantic-kernel', 'jinja2' or 'handlebars'.

Name	Description
input_variables Required*	The input variables for the prompt.
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.
execution_settings Required*	The execution settings for the prompt.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
name Required*	
description Required*	
template Required*	
template_format	default value: semantic-kernel
input_variables Required*	
allow_dangerously_set_content Required*	
execution_settings Required*	

Methods

[\[\] Expand table](#)

add_execution_settings	Add execution settings to the prompt template.
check_input_variables	Verify that input variable default values are string only

<code>from_json</code>	Create a PromptTemplateConfig instance from a JSON string.
<code>get_kernel_parameter_metadata</code>	Get the kernel parameter metadata for the input variables.
<code>restore</code>	Restore a PromptTemplateConfig instance from the specified parameters.
<code>rewrite_execution_settings</code>	Rewrite execution settings to a dictionary.

add_execution_settings

Add execution settings to the prompt template.

Python

```
add_execution_settings(settings: PromptExecutionSettings, overwrite: bool = True) -> None
```

Parameters

[+] Expand table

Name	Description
<code>settings</code> Required*	
<code>overwrite</code>	default value: True

check_input_variables

Verify that input variable default values are string only

Python

```
check_input_variables()
```

from_json

Create a PromptTemplateConfig instance from a JSON string.

Python

```
from_json(json_str: str) -> PromptTemplateConfig
```

Parameters

[+] Expand table

Name	Description
json_str Required*	

get_kernel_parameter_metadata

Get the kernel parameter metadata for the input variables.

Python

```
get_kernel_parameter_metadata() ->
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterM
etadata]
```

restore

Restore a PromptTemplateConfig instance from the specified parameters.

Python

```
restore(name: str, description: str, template: str, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
input_variables:
list[semantic_kernel.prompt_template.input_variable.InputVariable] = [],
execution_settings: dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe
ttings] = {}, allow_dangerously_set_content: bool = False) ->
PromptTemplateConfig
```

Parameters

[+] Expand table

Name	Description
name Required*	The name of the prompt template.
description Required*	The description of the prompt template.
template Required*	The template for the prompt.
input_variables	The input variables for the prompt. default value: []
execution_settings	The execution settings for the prompt. default value: {}
template_format	default value: semantic-kernel
allow_dangerously_set_content	default value: False

Returns

[] Expand table

Type	Description
	A new PromptTemplateConfig instance.

rewrite_execution_settings

Rewrite execution settings to a dictionary.

Python

```
rewrite_execution_settings(settings: None | PromptExecutionSettings |
list[semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings] | dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings]) -> dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings]
```

Parameters

Name	Description
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =
{'allow_dangerously_set_content': FieldInfo(annotation=bool,
required=False, default=False), 'description':
FieldInfo(annotation=Union[str, NoneType], required=False, default=''),
'execution_settings': FieldInfo(annotation=dict[str,
PromptExecutionSettings], required=False, default_factory=dict),
```

```
'input_variables': FieldInfo(annotation=list[InputVariable],  
required=False, default_factory=list), 'name': FieldInfo(annotation=str,  
required=False, default=''), 'template': FieldInfo(annotation=Union[str,  
NoneType], required=False, default=None), 'template_format':  
FieldInfo(annotation=Literal['semantic-kernel', 'handlebars', 'jinja2'],  
required=False, default='semantic-kernel')}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

description

Python

```
description: str | None
```

execution_settings

Python

```
execution_settings: dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

input_variables

Python

```
input_variables:  
list[semantic_kernel.prompt_template.input_variable.InputVariable]
```

name

Python

```
name: str
```

template

Python

```
template: str | None
```

template_format

Python

```
template_format: Literal['semantic-kernel', 'handlebars', 'jinja2']
```

HandlebarsPromptTemplate Class

Reference

Create a Handlebars prompt template.

Handlebars are parsed as a whole and therefore do not have variables that can be extracted, also with handlebars there is no distinction in syntax between a variable and a value, a value that is encountered is tried to resolve with the arguments and the functions, if not found, the literal value is returned.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [PromptTemplateBase](#) → HandlebarsPromptTemplate

Constructor

Python

```
HandlebarsPromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

[\[\]](#) Expand table

Name	Description
<code>prompt_template_config</code> Required*	<code>PromptTemplateConfig</code> The prompt template configuration This is checked if the template format is 'handlebars'
<code>allow_dangerously_set_content</code> Required*	<code><xref:<xref:semantic_kernel.prompt_template.bool = False>></code> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>prompt_template_config</code> Required*	
<code>allow_dangerously_set_content</code> Required*	

Methods

[+] Expand table

<code>model_post_init</code>	We need to both initialize private attributes and call the user-defined <code>model_post_init</code> method.
<code>render</code>	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
<code>validate_template_format</code>	

`model_post_init`

We need to both initialize private attributes and call the user-defined `model_post_init` method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[+] Expand table

Name	Description
<code>_ModelMetaclass__context</code> Required*	

render

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) ->  
str
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	The kernel instance
arguments	The kernel arguments default value: None

Returns

[\[\] Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[\[\] Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =
{'allow_dangerously_set_content': FieldInfo(annotation=bool,
required=False, default=False), 'prompt_template_config':
FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

InputVariable Class

Reference

Input variable for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `InputVariable`

Constructor

Python

```
InputVariable(*, name: str, description: str | None = '', default: Any | None = None, is_required: bool | None = True, json_schema: str | None = None, allow_dangerously_set_content: bool = False)
```

Parameters

[+] [Expand table](#)

Name	Description
<code>name</code> Required*	The name of the input variable.
<code>description</code> Required*	The description of the input variable.
<code>default</code> Required*	The default value of the input variable.
<code>is_required</code> Required*	Whether the input variable is required.
<code>json_schema</code> Required*	The JSON schema for the input variable.

Name	Description
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
name Required*	
description Required*	
default Required*	
is_required	default value: True
json_schema Required*	
allow_dangerously_set_content Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
'allow_dangerously_set_content': FieldInfo(annotation=bool,  
required=False, default=False), 'default':  
FieldInfo(annotation=Union[Any, NoneType], required=False, default=''),  
'description': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=''), 'is_required': FieldInfo(annotation=Union[bool, NoneType],  
required=False, default=True), 'json_schema':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=''),  
'name': FieldInfo(annotation=str, required=True)}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

default

Python

```
default: Any | None
```

description

Python

```
description: str | None
```

is_required

Python

```
is_required: bool | None
```

json_schema

Python

```
json_schema: str | None
```

name

Python

```
name: str
```

Jinja2PromptTemplate Class

Reference

Creates and renders Jinja2 prompt templates to text.

Jinja2 templates support advanced features such as variable substitution, control structures, and inheritance, making it possible to dynamically generate text based on input arguments and predefined functions. This class leverages Jinja2's flexibility to render prompts that can include conditional logic, loops, and functions, based on the provided template configuration and arguments.

Note that the fully qualified function name (in the form of "plugin-function") is not allowed in Jinja2 because of the hyphen. Therefore, the function name is replaced with an underscore, which are allowed in Python function names.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `PromptTemplateBase` → `Jinja2PromptTemplate`

Constructor

Python

```
Jinja2PromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

 Expand table

Name	Description
<code>prompt_template_config</code> Required*	<code>PromptTemplateConfig</code> The configuration object for the prompt template. This should specify the template format as 'jinja2' and include any necessary configuration details required for rendering the template.

Name	Description
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
prompt_template_config Required*	
allow_dangerously_set_content Required*	

Methods

[\[\] Expand table](#)

model_post_init	We need to both initialize private attributes and call the user-defined model_post_init method.
render	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
validate_template_format	

[model_post_init](#)

We need to both initialize private attributes and call the user-defined model_post_init method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>_ModelMetaclass__context</code> Required*	

render

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) ->
str
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>kernel</code> Required*	The kernel instance
<code>arguments</code>	The kernel arguments default value: None

Returns

[\[\] Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[\[\] Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'allow_dangerously_set_content': FieldInfo(annotation=bool,
```

```
required=False, default=False), 'prompt_template_config':  
    FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

KernelPromptTemplate Class

Reference

Create a Kernel prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [PromptTemplateBase](#) → KernelPromptTemplate

Constructor

Python

```
KernelPromptTemplate(*, prompt_template_config: PromptTemplateConfig,  
allow_dangerously_set_content: bool = False)
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	PromptTemplateConfig The prompt template configuration This includes the actual template to use.
<code>allow_dangerously_set_content</code> Required*	<code><xref:<xref:semantic_kernel.prompt_template.bool = False>></code> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
<code>prompt_template_config</code> Required*	
<code>allow_dangerously_set_content</code> Required*	

Methods

[\[\] Expand table](#)

<code>extract_blocks</code>	Given a prompt template string, extract all the blocks (text, variables, function calls).
<code>model_post_init</code>	We need to both initialize private attributes and call the user-defined <code>model_post_init</code> method.
<code>render</code>	Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.
<code>render_blocks</code>	Given a list of blocks render each block and compose the final result.
<code>validate_template_format</code>	

`extract_blocks`

Given a prompt template string, extract all the blocks (text, variables, function calls).

Python

```
extract_blocks() ->
list[semantic_kernel.template_engine.blocks.block.Block]
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>template_text</code> Required*	Prompt template

Returns

[\[\] Expand table](#)

Type	Description
	A list of all the blocks, ie the template tokenized in text, variables and function calls

model_post_init

We need to both initialize private attributes and call the user-defined model_post_init method.

Python

```
model_post_init(_ModelMetaclass__context: Any) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
_ModelMetaclass__context Required*	

render

Using the prompt template, replace the variables with their values and execute the functions replacing their reference with the function result.

Python

```
async render(kernel: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	The kernel instance
arguments	The kernel arguments default value: None

Returns

[\[\] Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

render_blocks

Given a list of blocks render each block and compose the final result.

Python

```
async render_blocks:
    list[semantic_kernel.template_engine.blocks.block.Block], kernel: Kernel,
    arguments: KernelArguments) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
blocks Required*	Template blocks generated by ExtractBlocks
context Required*	Access into the current kernel execution context
kernel Required*	
arguments Required*	

Returns

[\[\] Expand table](#)

Type	Description
	The prompt template ready to be used for an AI request

validate_template_format

Python

```
validate_template_format(v: PromptTemplateConfig) -> PromptTemplateConfig
```

Parameters

[\[\] Expand table](#)

Name	Description
v Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'allow_dangerously_set_content': FieldInfo(annotation=bool,  
required=False, default=False), 'prompt_template_config':  
FieldInfo(annotation=PromptTemplateConfig, required=True)}
```

PromptTemplateConfig Class

Reference

Configuration for a prompt template.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `PromptTemplateConfig`

Constructor

Python

```
PromptTemplateConfig(*, name: str = '', description: str | None = '',
template: str | None = None, template_format: Literal['semantic-kernel',
'handlebars', 'jinja2'] = 'semantic-kernel', input_variables:
list[semantic_kernel.prompt_template.input_variable.InputVariable] = None,
allow_dangerously_set_content: bool = False, execution_settings: dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSetti
ngs] = None)
```

Parameters

[] Expand table

Name	Description
name Required*	The name of the prompt template.
description Required*	The description of the prompt template.
template Required*	The template for the prompt.
template_format Required*	The format of the template, should be 'semantic-kernel', 'jinja2' or 'handlebars'.

Name	Description
input_variables Required*	The input variables for the prompt.
allow_dangerously_set_content Required*	<xref:<xref:semantic_kernel.prompt_template.bool = False>> Allow content without encoding throughout, this overrides the same settings in the prompt template config and input variables. This reverts the behavior to unencoded input.
execution_settings Required*	The execution settings for the prompt.

Keyword-Only Parameters

[+] Expand table

Name	Description
name Required*	
description Required*	
template Required*	
template_format	default value: semantic-kernel
input_variables Required*	
allow_dangerously_set_content Required*	
execution_settings Required*	

Methods

[+] Expand table

add_execution_settings	Add execution settings to the prompt template.
check_input_variables	Verify that input variable default values are string only

<code>from_json</code>	Create a PromptTemplateConfig instance from a JSON string.
<code>get_kernel_parameter_metadata</code>	Get the kernel parameter metadata for the input variables.
<code>restore</code>	Restore a PromptTemplateConfig instance from the specified parameters.
<code>rewrite_execution_settings</code>	Rewrite execution settings to a dictionary.

add_execution_settings

Add execution settings to the prompt template.

Python

```
add_execution_settings(settings: PromptExecutionSettings, overwrite: bool = True) -> None
```

Parameters

[+] Expand table

Name	Description
<code>settings</code> Required*	
<code>overwrite</code>	default value: True

check_input_variables

Verify that input variable default values are string only

Python

```
check_input_variables()
```

from_json

Create a PromptTemplateConfig instance from a JSON string.

Python

```
from_json(json_str: str) -> PromptTemplateConfig
```

Parameters

[+] Expand table

Name	Description
json_str Required*	

get_kernel_parameter_metadata

Get the kernel parameter metadata for the input variables.

Python

```
get_kernel_parameter_metadata() ->
list[semantic_kernel.functions.kernel_parameter_metadata.KernelParameterM
etadata]
```

restore

Restore a PromptTemplateConfig instance from the specified parameters.

Python

```
restore(name: str, description: str, template: str, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
input_variables:
list[semantic_kernel.prompt_template.input_variable.InputVariable] = [],
execution_settings: dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe
ttings] = {}, allow_dangerously_set_content: bool = False) ->
PromptTemplateConfig
```

Parameters

[+] Expand table

Name	Description
name Required*	The name of the prompt template.
description Required*	The description of the prompt template.
template Required*	The template for the prompt.
input_variables	The input variables for the prompt. default value: []
execution_settings	The execution settings for the prompt. default value: {}
template_format	default value: semantic-kernel
allow_dangerously_set_content	default value: False

Returns

[] Expand table

Type	Description
	A new PromptTemplateConfig instance.

rewrite_execution_settings

Rewrite execution settings to a dictionary.

Python

```
rewrite_execution_settings(settings: None | PromptExecutionSettings |
list[semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings] | dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings]) -> dict[str,
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings]
```

Parameters

Name	Description
settings Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =
{'allow_dangerously_set_content': FieldInfo(annotation=bool,
required=False, default=False), 'description':
FieldInfo(annotation=Union[str, NoneType], required=False, default=''),
'execution_settings': FieldInfo(annotation=dict[str,
PromptExecutionSettings], required=False, default_factory=dict),
```

```
'input_variables': FieldInfo(annotation=list[InputVariable],  
required=False, default_factory=list), 'name': FieldInfo(annotation=str,  
required=False, default=''), 'template': FieldInfo(annotation=Union[str,  
NoneType], required=False, default=None), 'template_format':  
FieldInfo(annotation=Literal['semantic-kernel', 'handlebars', 'jinja2'],  
required=False, default='semantic-kernel')}
```

allow_dangerously_set_content

Python

```
allow_dangerously_set_content: bool
```

description

Python

```
description: str | None
```

execution_settings

Python

```
execution_settings: dict[str,  
semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSe  
ttings]
```

input_variables

Python

```
input_variables:  
list[semantic_kernel.prompt_template.input_variable.InputVariable]
```

name

Python

```
name: str
```

template

Python

```
template: str | None
```

template_format

Python

```
template_format: Literal['semantic-kernel', 'handlebars', 'jinja2']
```

services Package

Reference

Modules

[\[\] Expand table](#)

[ai_service_client_base](#)

[ai_service_selector](#)

[kernel_services_extension](#)

Classes

[\[\] Expand table](#)

[AIServiceSelector](#) Default service selector, can be subclassed and overridden.

To use a custom service selector, subclass this class and override the select_ai_service method. Make sure that the function signature stays the same.

ai_service_client_base Module

Reference

Classes

[\[\] Expand table](#)

[AIServiceClientBase](#) Base class for all AI Services.

Has a ai_model_id and service_id, any other fields have to be defined by the subclasses.

The ai_model_id can refer to a specific model, like 'gpt-35-turbo' for OpenAI, or can just be a string that is used to identify the model in the service.

The service_id is used in Semantic Kernel to identify the service, if empty the ai_model_id is used.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

AIServiceClientBase Class

Reference

Base class for all AI Services.

Has a ai_model_id and service_id, any other fields have to be defined by the subclasses.

The ai_model_id can refer to a specific model, like 'gpt-35-turbo' for OpenAI, or can just be a string that is used to identify the model in the service.

The service_id is used in Semantic Kernel to identify the service, if empty the ai_model_id is used.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → AIServiceClientBase

[ABC](#) → AIServiceClientBase

Constructor

Python

```
AIServiceClientBase(*, ai_model_id: str, service_id: str = '')
```

Keyword-Only Parameters

[\[+\] Expand table](#)

Name	Description
ai_model_id Required*	
service_id Required*	

Methods

[Expand table](#)

get_prompt_execution_settings_class	Get the request settings class.
get_prompt_execution_settings_from_settings	Get the request settings from a settings object.
instantiate_prompt_execution_settings	Create a request settings object. All arguments are passed to the constructor of the request settings object.
model_post_init	Update the service_id if it is not set.

get_prompt_execution_settings_class

Get the request settings class.

Python

```
get_prompt_execution_settings_class() -> PromptExecutionSettings
```

get_prompt_execution_settings_from_settings

Get the request settings from a settings object.

Python

```
get_prompt_execution_settings_from_settings(settings:  
    PromptExecutionSettings) -> PromptExecutionSettings
```

Parameters

[Expand table](#)

Name	Description
settings Required*	

instantiate_prompt_execution_settings

Create a request settings object.

All arguments are passed to the constructor of the request settings object.

Python

```
instantiate_prompt_execution_settings(**kwargs) ->
PromptExecutionSettings
```

model_post_init

Update the service_id if it is not set.

Python

```
model_post_init(_AIServiceClientBase__context: object | None = None)
```

Parameters

[+] Expand table

Name	Description
_AIServiceClientBase__context	default value: None

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_model_id':  
FieldInfo(annotation=str, required=True, metadata=  
[StringConstraints(strip_whitespace=True, to_upper=None, to_lower=None,  
strict=None, min_length=1, max_length=None, pattern=None)]),  
'service_id': FieldInfo(annotation=str, required=False, default='')}
```

ai_model_id

Python

```
ai_model_id: str
```

service_id

Python

```
service_id: str
```

ai_service_selector Module

Reference

Classes

[+] Expand table

AIServiceSelector	Default service selector, can be subclassed and overridden.
-----------------------------------	---

To use a custom service selector, subclass this class and override the select_ai_service method. Make sure that the function signature stays the same.

AIServiceSelector Class

Reference

Default service selector, can be subclassed and overridden.

To use a custom service selector, subclass this class and override the `select_ai_service` method. Make sure that the function signature stays the same.

Inheritance `builtins.object` → `AIServiceSelector`

Constructor

Python

```
AIServiceSelector()
```

Methods

[+] Expand table

<code>select_ai_service</code>	Select a AI Service on a first come, first served basis, starting with execution settings in the arguments, followed by the execution settings from the function. If the same <code>service_id</code> is in both, the one in the arguments will be used.
--------------------------------	--

select_ai_service

Select a AI Service on a first come, first served basis, starting with execution settings in the arguments, followed by the execution settings from the function. If the same `service_id` is in both, the one in the arguments will be used.

Python

```
select_ai_service(kernel: Kernel, function: KernelFunction, arguments: KernelArguments, type_: type['AI_SERVICE_CLIENT_TYPE'] | None = None) -> tuple['AI_SERVICE_CLIENT_TYPE', 'PromptExecutionSettings']
```

Parameters

[\[\]](#) Expand table

Name	Description
kernel Required*	
function Required*	
arguments Required*	
type_	default value: None

kernel_services_extension Module

Reference

Classes

[+] Expand table

<p><code>KernelServicesExtension</code></p>	<p>Create a new model by parsing and validating input data from keyword arguments.</p>
---	--

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

KernelServicesExtension Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `KernelServicesExtension`

`ABC` → `KernelServicesExtension`

Constructor

Python

```
KernelServicesExtension(*, services: dict[str,  
semantic_kernel.services.ai_service_client_base.AIServiceClientBase] = None,  
ai_service_selector: AIServiceSelector = None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>services</code> Required*	
<code>ai_service_selector</code> Required*	

Methods

[+] Expand table

<code>add_service</code>	
<code>get_prompt_execution_settings_from_service_id</code>	Get the specific request settings from the service, instantiated with the <code>service_id</code> and

	ai_model_id.
get_service	<p>Get a service by service_id and type.</p> <p>Type is optional and when not supplied, no checks are done. Type should be</p> <p>TextCompletionClientBase, ChatCompletionClientBase, EmbeddingGeneratorBase or a subclass of one. You can also check for multiple types in one go, by using TextCompletionClientBase ChatCompletionClientBase.</p> <p>If type and service_id are both None, the first service is returned.</p>
get_services_by_type	
remove_all_services	Removes the services from the Kernel, does not delete them.
remove_service	Delete a single service from the Kernel.
rewrite_services	Rewrite services to a dictionary.
select_ai_service	Uses the AI service selector to select a service for the function.

add_service

Python

```
add_service(service: AIServiceClientBase, overwrite: bool = False) ->
None
```

Parameters

[+] Expand table

Name	Description
service Required*	
overwrite	default value: False

get_prompt_execution_settings_from_service_id

Get the specific request settings from the service, instantiated with the service_id and ai_model_id.

Python

```
get_prompt_execution_settings_from_service_id(service_id: str, type:  
type[typing.Union[ForwardRef('TextCompletionClientBase'),  
ForwardRef('ChatCompletionClientBase'),  
ForwardRef('EmbeddingGeneratorBase')]] | None = None) ->  
PromptExecutionSettings
```

Parameters

[+] Expand table

Name	Description
service_id Required*	
type	default value: None

get_service

Get a service by service_id and type.

Type is optional and when not supplied, no checks are done. Type should be

TextCompletionClientBase, ChatCompletionClientBase, EmbeddingGeneratorBase or a subclass of one. You can also check for multiple types in one go, by using TextCompletionClientBase | ChatCompletionClientBase.

If type and service_id are both None, the first service is returned.

Python

```
get_service(service_id: str | None = None, type:  
type[typing.Union[ForwardRef('TextCompletionClientBase'),  
ForwardRef('ChatCompletionClientBase'),  
ForwardRef('EmbeddingGeneratorBase')]] | None = None) ->  
AIServiceClientBase
```

Parameters

[\[\] Expand table](#)

Name	Description
service_id	<xref:<xref:semantic_kernel.services.kernel_services_extension.str None>> The service id, if None, the default service is returned or the first service is returned. default value: None
type	<xref:Type>[<xref:ALL_SERVICE_TYPES>]<xref: None> The type of the service, if None, no checks are done. default value: None

Returns

[\[\] Expand table](#)

Type	Description
<xref:ALL_SERVICE_TYPES>	The service.

Exceptions

[\[\] Expand table](#)

Type	Description
ValueError	If no service is found that matches the type.

get_services_by_type

Python

```
get_services_by_type(type:  
type[typing.Union[ForwardRef('TextCompletionClientBase'),  
ForwardRef('ChatCompletionClientBase'),  
ForwardRef('EmbeddingGeneratorBase')]]) -> dict[str,  
typing.Union[ForwardRef('TextCompletionClientBase'),  
ForwardRef('ChatCompletionClientBase'),  
ForwardRef('EmbeddingGeneratorBase')]]
```

Parameters

[\[\] Expand table](#)

Name	Description
type Required*	

remove_all_services

Removes the services from the Kernel, does not delete them.

Python

```
remove_all_services() -> None
```

remove_service

Delete a single service from the Kernel.

Python

```
remove_service(service_id: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
service_id Required*	

rewrite_services

Rewrite services to a dictionary.

Python

```
rewrite_services(services: AI_SERVICE_CLIENT_TYPE |  
list[AI_SERVICE_CLIENT_TYPE] | dict[str, AI_SERVICE_CLIENT_TYPE] | None =
```

```
None) -> dict[str, AI_SERVICE_CLIENT_TYPE]
```

Parameters

[\[\] Expand table](#)

Name	Description
services	default value: None

select_ai_service

Uses the AI service selector to select a service for the function.

Python

```
select_ai_service(function: KernelFunction, arguments: KernelArguments) -> tuple[typing.Union[ForwardRef('TextCompletionClientBase'), ForwardRef('ChatCompletionClientBase'), ForwardRef('EmbeddingGeneratorBase')], semantic_kernel.connectors.ai.prompt_execution_settings.PromptExecutionSettings]
```

Parameters

[\[\] Expand table](#)

Name	Description
function Required*	
arguments Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_service_selector':  
FieldInfo(annotation=AIServiceSelector, required=False,  
default_factory=AIServiceSelector), 'services':  
FieldInfo(annotation=dict[str, AIServiceClientBase], required=False,  
default_factory=dict)}
```

ai_service_selector

Python

```
ai_service_selector: AIServiceSelector
```

services

Python

```
services: dict[str,  
semantic_kernel.services.ai_service_client_base.AIServiceClientBase]
```

AIServiceSelector Class

Reference

Default service selector, can be subclassed and overridden.

To use a custom service selector, subclass this class and override the `select_ai_service` method. Make sure that the function signature stays the same.

Inheritance `builtins.object` → `AIServiceSelector`

Constructor

Python

```
AIServiceSelector()
```

Methods

[+] Expand table

<code>select_ai_service</code>	Select a AI Service on a first come, first served basis, starting with execution settings in the arguments, followed by the execution settings from the function. If the same <code>service_id</code> is in both, the one in the arguments will be used.
--------------------------------	--

`select_ai_service`

Select a AI Service on a first come, first served basis, starting with execution settings in the arguments, followed by the execution settings from the function. If the same `service_id` is in both, the one in the arguments will be used.

Python

```
select_ai_service(kernel: Kernel, function: KernelFunction, arguments: KernelArguments, type_: type['AI_SERVICE_CLIENT_TYPE'] | None = None) -> tuple['AI_SERVICE_CLIENT_TYPE', 'PromptExecutionSettings']
```

Parameters

[\[\]](#) Expand table

Name	Description
kernel Required*	
function Required*	
arguments Required*	
type_	default value: None

text Package

Reference

Modules

 Expand table

function_extension	
<code>text_chunker</code>	Split text in chunks, attempting to leave meaning intact. For plain text, split looking at new lines first, then periods, and so on. For markdown, split looking at punctuation first, and so on.

Functions

aggregate_chunked_results

Aggregate the results from the chunked results.

Python

```
async aggregate_chunked_results(func: KernelFunction, chunked_results: list[str], kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

 Expand table

Name	Description
<code>func</code> Required*	
<code>chunked_results</code> Required*	
<code>kernel</code> Required*	
<code>arguments</code> Required*	

split_markdown_lines

Split markdown into lines. It will split on punctuation first, and then on space and new lines.

Python

```
split_markdown_lines(text: str, max_token_per_line: int, token_counter: ~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[+] [Expand table](#)

Name	Description
text Required*	
max_token_per_line Required*	
token_counter	

split_markdown_paragraph

Split markdown into paragraphs.

Python

```
split_markdown_paragraph(text: list[str], max_tokens: int, token_counter: ~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[+] [Expand table](#)

Name	Description
text Required*	
max_tokens Required*	

Name	Description
<code>token_counter</code>	

split_plaintext_lines

Split plain text into lines. it will split on new lines first, and then on punctuation.

Python

```
split_plaintext_lines(text: str, max_token_per_line: int, token_counter:  
~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[\[\]](#) Expand table

Name	Description
<code>text</code> Required*	
<code>max_token_per_line</code> Required*	
<code>token_counter</code>	

split_plaintext_paragraph

Split plain text into paragraphs.

Python

```
split_plaintext_paragraph(text: list[str], max_tokens: int,  
token_counter: ~collections.abc.Callable = <function _token_counter>) ->  
list[str]
```

Parameters

[\[\]](#) Expand table

Name	Description
text Required*	
max_tokens Required*	
token_counter	

function_extension Module

Reference

Functions

aggregate_chunked_results

Aggregate the results from the chunked results.

Python

```
async aggregate_chunked_results(func: KernelFunction, chunked_results: list[str], kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
func Required*	
chunked_results Required*	
kernel Required*	
arguments Required*	

text_chunker Module

Reference

Split text in chunks, attempting to leave meaning intact. For plain text, split looking at new lines first, then periods, and so on. For markdown, split looking at punctuation first, and so on.

Functions

split_markdown_lines

Split markdown into lines. It will split on punctuation first, and then on space and new lines.

Python

```
split_markdown_lines(text: str, max_token_per_line: int, token_counter:  
~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[] Expand table

Name	Description
text Required*	
max_token_per_line Required*	
token_counter	

split_markdown_paragraph

Split markdown into paragraphs.

Python

```
split_markdown_paragraph(text: list[str], max_tokens: int, token_counter:
```

```
~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
text Required*	
max_tokens Required*	
token_counter	

split_plaintext_lines

Split plain text into lines. it will split on new lines first, and then on punctuation.

Python

```
split_plaintext_lines(text: str, max_token_per_line: int, token_counter:  
~collections.abc.Callable = <function _token_counter>) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
text Required*	
max_token_per_line Required*	
token_counter	

split_plaintext_paragraph

Split plain text into paragraphs.

Python

```
split_plaintext_paragraph(text: list[str], max_tokens: int,  
token_counter: ~collections.abc.Callable = <function _token_counter>) ->  
list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
text Required*	
max_tokens Required*	
token_counter	

const Module

Reference

conversation_summary_plugin Module

Reference

Classes

 [Expand table](#)

ConversationSummaryPlugin	Semantic plugin that enables conversations summarization.
---	---

Initializes a new instance of the ConversationSummaryPlugin class.

ConversationSummaryPlugin Class

Reference

Semantic plugin that enables conversations summarization.

Initializes a new instance of the ConversationSummaryPlugin class.

Inheritance builtins.object → ConversationSummaryPlugin

Constructor

Python

```
ConversationSummaryPlugin(kernel: Kernel, prompt_template_config:  
PromptTemplateConfig, return_key: str = 'summary')
```

Parameters

[] Expand table

Name	Description
kernel Required*	The kernel instance.
prompt_template_config Required*	The prompt template configuration.
return_key	The key to use for the return value. default value: summary

Methods

[] Expand table

kernel_function	Decorator for kernel functions, can be used directly as @kernel_function or with parameters @kernel_function(name='function', description='I am a function.'). This decorator is used to mark a function as a kernel function. It also provides metadata for the function. The name and description can be left empty, and then the function name and docstring will be used.
---------------------------------	--

The parameters are parsed from the function signature, use `typing.Annotated` to provide a description for the parameter, in python 3.8, use `typing_extensions.Annotated`.

To parse the type, first it checks if the parameter is annotated, and get's the description from there. After that it checks recursively until it reaches the lowest level, and it combines the types into a single comma-separated string, a `forwardRef` is also supported. All of this is are stored in `kernel_function_parameters`.

The return type and description are parsed from the function signature, and that is stored in `kernel_function_return_type`, `kernel_function_return_description` and `kernel_function_return_required`.

It also checks if the function is a streaming type (generator or iterable, `async` or not), and that is stored as a bool in `kernel_function_streaming`.

`summarize_conversation` Given a long conversation transcript, summarize the conversation.

kernel_function

Decorator for kernel functions, can be used directly as `@kernel_function` or with parameters `@kernel_function(name='function', description='I am a function.')`.

This decorator is used to mark a function as a kernel function. It also provides metadata for the function. The name and description can be left empty, and then the function name and docstring will be used.

The parameters are parsed from the function signature, use `typing.Annotated` to provide a description for the parameter, in python 3.8, use `typing_extensions.Annotated`.

To parse the type, first it checks if the parameter is annotated, and get's the description from there. After that it checks recursively until it reaches the lowest level, and it combines the types into a single comma-separated string, a `forwardRef` is also supported. All of this is are stored in `kernel_function_parameters`.

The return type and description are parsed from the function signature, and that is stored in `kernel_function_return_type`, `kernel_function_return_description` and `kernel_function_return_required`.

It also checks if the function is a streaming type (generator or iterable, `async` or not), and that is stored as a bool in `kernel_function_streaming`.

Python

```
kernel_function(name: str | None = None, description: str | None = None)
-> Callable[..., Any]
```

Parameters

[+] Expand table

Name	Description
name	<xref:<xref:semantic_kernel.core_plugins.conversation_summary_plugin.str None>> default value: None
description	<xref:<xref:semantic_kernel.core_plugins.conversation_summary_plugin.str None>> if not supplied, the function docstring will be used, can be None. default value: None
func	default value: None

summarize_conversation

Given a long conversation transcript, summarize the conversation.

Python

```
async summarize_conversation(input: str, kernel: Kernel, arguments:
KernelArguments) -> KernelArguments
```

Parameters

[+] Expand table

Name	Description
input Required*	A long conversation transcript.
kernel Required*	The kernel for function execution.
arguments	Arguments used by the kernel.

Name	Description
Required*	

Returns

[\[\] Expand table](#)

Type	Description
	KernelArguments with the summarized conversation result in key self.return_key.

auto_function_invocation_context

Module

Reference

Classes

[] [Expand table](#)

[AutoFunctionInvocationContext](#) Class for auto function invocation context.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

AutoFunctionInvocationContext Class

Reference

Class for auto function invocation context.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [FilterContextBase](#) → AutoFunctionInvocationContext

Constructor

Python

```
AutoFunctionInvocationContext()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'arguments':  
    FieldInfo(annotation=ForwardRef('KernelArguments'), required=True),  
    'chat_history': FieldInfo(annotation=ForwardRef('ChatHistory | None'),  
        required=False, default=None), 'function':  
    FieldInfo(annotation=ForwardRef('KernelFunction'), required=True),  
    'function_count': FieldInfo(annotation=int, required=False, default=0),  
    'function_result': FieldInfo(annotation=ForwardRef('FunctionResult |  
None'), required=False, default=None), 'function_sequence_index':  
    FieldInfo(annotation=int, required=False, default=0), 'kernel':  
    FieldInfo(annotation=ForwardRef('Kernel'), required=True),  
    'request_sequence_index': FieldInfo(annotation=int, required=False,  
        default=0), 'terminate': FieldInfo(annotation=bool, required=False,  
        default=False)}
```

chat_history

Python

```
chat_history: ChatHistory | None
```

function_count

Python

```
function_count: int
```

function_result

Python

```
function_result: FunctionResult | None
```

function_sequence_index

Python

```
function_sequence_index: int
```

request_sequence_index

Python

```
request_sequence_index: int
```

terminate

Python

```
terminate: bool
```

filter_context_base Module

Reference

Classes

 [Expand table](#)

[FilterContextBase](#) Base class for Kernel Filter Contexts.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FilterContextBase Class

Reference

Base class for Kernel Filter Contexts.

Note: This class is experimental and may change in the future.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → FilterContextBase

Constructor

Python

```
FilterContextBase()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'arguments':  
FieldInfo(annotation=ForwardRef('KernelArguments'), required=True),  
'function': FieldInfo(annotation=ForwardRef('KernelFunction'),  
required=True), 'kernel': FieldInfo(annotation=ForwardRef('Kernel'),  
required=True)}
```

arguments

Python

```
arguments: KernelArguments
```

function

Python

```
function: KernelFunction
```

is_experimental

Python

```
is_experimental = True
```

kernel

Python

kernel: Kernel

filter_types Module

Reference

Enums

[\[\] Expand table](#)

FilterTypes	Enum for the filter types.
-----------------------------	----------------------------

Note: This class is experimental and may change in the future.

FilterTypes Enum

Reference

Enum for the filter types.

Note: This class is experimental and may change in the future.

Inheritance builtins.str → FilterTypes

[Enum](#) → FilterTypes

Constructor

Python

```
FilterTypes(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[\[\] Expand table](#)

AUTO_FUNCTION_INVOCATION
FUNCTION_INVOCATION
PROMPT_RENDERING
is_experimental

function_invocation_context Module

Reference

Classes

[+] Expand table

[FunctionInvocationContext](#) Class for function invocation context.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FunctionInvocationContext Class

Reference

Class for function invocation context.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [FilterContextBase](#) → FunctionInvocationContext

Constructor

Python

```
FunctionInvocationContext()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'arguments':  
    FieldInfo(annotation=ForwardRef('KernelArguments'), required=True),  
    'function': FieldInfo(annotation=ForwardRef('KernelFunction'),  
    required=True), 'kernel': FieldInfo(annotation=ForwardRef('Kernel'),  
    required=True), 'result': FieldInfo(annotation=ForwardRef('FunctionResult  
    | None'), required=False, default=None)}
```

result

Python

```
result: FunctionResult | None
```

kernel_filters_extension Module

Reference

Classes

[\[\] Expand table](#)

[KernelFilterExtension](#) KernelFilterExtension.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

KernelFilterExtension Class

Reference

KernelFilterExtension.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → KernelFilterExtension

[ABC](#) → KernelFilterExtension

Constructor

Python

```
KernelFilterExtension(*, function_invocation_filters: list[tuple[int,
collections.abc.Callable[[FILTER_CONTEXT_TYPE,
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]] = None,
prompt_rendering_filters: list[tuple[int,
collections.abc.Callable[[FILTER_CONTEXT_TYPE,
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]] = None,
auto_function_invocation_filters: list[tuple[int,
collections.abc.Callable[[FILTER_CONTEXT_TYPE,
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]] = None)
```

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
function_invocation_filters Required*	
prompt_rendering_filters Required*	
auto_function_invocation_filters Required*	

Methods

[Expand table](#)

add_filter	<p>Add a filter to the Kernel.</p> <p>Each filter is added to the beginning of the list of filters, this is because the filters are executed in the order they are added, so the first filter added, will be the first to be executed, but it will also be the last executed for the part after <i>await next(context)</i>.</p> <p>Args: filter_type (str): The type of the filter to add (function_invocation, prompt_rendering) filter (object): The filter to add</p> <p>Note: This function is experimental and may change in the future.</p>
construct_call_stack	
filter	<p>Decorator to add a filter to the Kernel.</p> <p>Note: This function is experimental and may change in the future.</p>

add_filter

Add a filter to the Kernel.

Each filter is added to the beginning of the list of filters, this is because the filters are executed in the order they are added, so the first filter added, will be the first to be executed, but it will also be the last executed for the part after *await next(context)*.

Args: filter_type (str): The type of the filter to add (function_invocation, prompt_rendering) filter (object): The filter to add

Note: This function is experimental and may change in the future.

Python

```

add_filter(filter_type:
~typing.Literal[<FilterTypes.AUTO_FUNCTION_INVOCATION:
'auto_function_invocation', <FilterTypes.FUNCTION_INVOCATION:
'function_invocation', <FilterTypes.PROMPT_RENDERING:
'prompt_rendering'] | ~semantic_kernel.filters.filter_types.FilterTypes,
filter:
~collections.abc.Callable[[~semantic_kernel.filters.kernel_filters_extens
ion.FILTER_CONTEXT_TYPE,
~collections.abc.Callable[[~semantic_kernel.filters.kernel_filters_extens
ion.FILTER_CONTEXT_TYPE], None]], None]) -> None

```

Parameters

[] Expand table

Name	Description
filter_type Required*	
filter Required*	

construct_call_stack

Python

```

construct_call_stack(filter_type: FilterTypes, inner_function:
Callable[[FILTER_CONTEXT_TYPE], Coroutine[Any, Any, None]]) ->
Callable[[FILTER_CONTEXT_TYPE], Coroutine[Any, Any, None]]

```

Parameters

[] Expand table

Name	Description
filter_type Required*	
inner_function Required*	

filter

Decorator to add a filter to the Kernel.

Note: This function is experimental and may change in the future.

Python

```
filter(filter_type:  
~typing.Literal[<FilterTypes.AUTO_FUNCTION_INVOCATION:  
'auto_function_invocation'>, <FilterTypes.FUNCTION_INVOCATION:  
'function_invocation'>, <FilterTypes.PROMPT_RENDERING:  
'prompt_rendering'>] | ~semantic_kernel.filters.filter_types.FilterTypes)  
-> Callable[[Callable[[FILTER_CONTEXT_TYPE,  
Callable[[FILTER_CONTEXT_TYPE], None]], None]],  
Callable[[FILTER_CONTEXT_TYPE, Callable[[FILTER_CONTEXT_TYPE], None]],  
None]]
```

Parameters

[\[+\]](#) Expand table

Name	Description
filter_type Required*	

remove_filter

Remove a filter from the Kernel.

Args: filter_type (str | FilterTypes | None): The type of the filter to remove.

```
filter_id (int): The id of the hook to remove  
position (int): The position of the filter in the list
```

Note: This function is experimental and may change in the future.

Python

```
remove_filter(filter_type:  
~typing.Literal[<FilterTypes.AUTO_FUNCTION_INVOCATION:  
'auto_function_invocation'>, <FilterTypes.FUNCTION_INVOCATION:  
'function_invocation'>, <FilterTypes.PROMPT_RENDERING:
```

```
'prompt_rendering' >] | ~semantic_kernel.filters.filter_types.FilterTypes  
| None = None, filter_id: int | None = None, position: int | None = None)  
-> None
```

Parameters

[+] Expand table

Name	Description
<code>filter_type</code>	default value: None
<code>filter_id</code>	default value: None
<code>position</code>	default value: None

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [`ConfigDict`] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [`FieldInfo`][`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] =  
{'auto_function_invocation_filters': FieldInfo(annotation=list[tuple[int,  
Callable[list, NoneType]]], required=False, default_factory=list),  
'function_invocation_filters': FieldInfo(annotation=list[tuple[int,  
Callable[list, NoneType]]], required=False, default_factory=list),  
'prompt_rendering_filters': FieldInfo(annotation=list[tuple[int,  
Callable[list, NoneType]]], required=False, default_factory=list)}
```

auto_function_invocation_filters

Python

```
auto_function_invocation_filters: list[tuple[int,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]]
```

function_invocation_filters

Python

```
function_invocation_filters: list[tuple[int,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]]
```

prompt_rendering_filters

Python

```
prompt_rendering_filters: list[tuple[int,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None]], None]]]
```

prompt_render_context Module

Reference

Classes

[+] Expand table

[PromptRenderContext](#) Context for prompt rendering filters.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

PromptRenderContext Class

Reference

Context for prompt rendering filters.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [FilterContextBase](#) → PromptRenderContext

Constructor

Python

```
PromptRenderContext()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'arguments':  
    FieldInfo(annotation=ForwardRef('KernelArguments'), required=True),  
    'function': FieldInfo(annotation=ForwardRef('KernelFunction'),  
    required=True), 'function_result':  
    FieldInfo(annotation=ForwardRef('FunctionResult | None'), required=False,  
    default=None), 'kernel': FieldInfo(annotation=ForwardRef('Kernel'),  
    required=True), 'rendered_prompt': FieldInfo(annotation=Union[str,  
    NoneType], required=False, default=None)}
```

function_result

Python

```
function_result: FunctionResult | None
```

rendered_prompt

Python

```
rendered_prompt: str | None
```

kernel Module

Reference

Classes

[Expand table](#)

Kernel The Kernel class is the main entry point for the Semantic Kernel. It provides the ability to run semantic/native functions, and manage plugins, memory, and AI services.

Initialize a new instance of the Kernel class.

Kernel Class

Reference

The Kernel class is the main entry point for the Semantic Kernel. It provides the ability to run semantic/native functions, and manage plugins, memory, and AI services.

Initialize a new instance of the Kernel class.

Inheritance [KernelFilterExtension](#) → Kernel
[KernelFunctionExtension](#) → Kernel
[KernelServicesExtension](#) → Kernel
[KernelReliabilityExtension](#) → Kernel

Constructor

Python

```
Kernel(plugins: KernelPlugin | dict[str,  
semantic_kernel.functions.kernel_plugin.KernelPlugin] |  
list[semantic_kernel.functions.kernel_plugin.KernelPlugin] | None = None,  
services: AI_SERVICE_CLIENT_TYPE | list[AI_SERVICE_CLIENT_TYPE] | dict[str,  
AI_SERVICE_CLIENT_TYPE] | None = None, ai_service_selector:  
AIServiceSelector | None = None, *, retry_mechanism: RetryMechanismBase =  
None, function_invocation_filters: list[tuple[int,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None\]\], None\\]\\]\\] = None,  
prompt\\_rendering\\_filters: list\\[tuple\\[int,  
collections.abc.Callable\\[\\[FILTER\\_CONTEXT\\_TYPE,  
collections.abc.Callable\\[\\[FILTER\\_CONTEXT\\_TYPE\\], None\\\]\\\], None\\\\]\\\\]\\\\] = None,  
auto\\\\_function\\\\_invocation\\\\_filters: list\\\\[tuple\\\\[int,  
collections.abc.Callable\\\\[\\\\[FILTER\\\\_CONTEXT\\\\_TYPE,  
collections.abc.Callable\\\\[\\\\[FILTER\\\\_CONTEXT\\\\_TYPE\\\\], None\\\\\]\\\\\], None\\\\\\]\\\\\\]\\\\\\] = None\\\\\\)
```

Parameters

[] [Expand table](#)

Name	Description
plugins	<xref:KernelPlugin dict> [str ,<xref: KernelPlugin>]<xref: list>[<xref:KernelPlugin>]<xref: None > The plugins to be used by the kernel, will be rewritten to a dict with plugin name as key

Name	Description
	default value: None
dict[str Required*	str (<xref:services> (AIServiceClientBase list[AIServiceClientBase]) The services to be used by the kernel, will be rewritten to a dict with service_id as key
None Required*	AIServiceClientBase] The services to be used by the kernel, will be rewritten to a dict with service_id as key
ai_service_selector	<xref:<xref:semantic_kernel.kernel.AIServiceSelector None>> The AI service selector to be used by the kernel, default is based on order of execution settings. default value: None
**kwargs Required*	Any Additional fields to be passed to the Kernel model, these are limited to retry_mechanism and function_invoking_handlers and function_invoked_handlers, the best way to add function_invoking_handlers and function_invoked_handlers is to use the add_function_invoking_handler and add_function_invoked_handler methods.
services	default value: None

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
retry_mechanism Required*	
function_invocation_filters Required*	
prompt_rendering_filters Required*	
auto_function_invocation_filters Required*	

Methods

invoke	<p>Execute one or more functions.</p> <p>When multiple functions are passed the FunctionResult of each is put into a list.</p> <pre>:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.kernel.str None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.kernel.str None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.kernel.dict[str, Any]> :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.kernel.dict[str, Any]></pre>
invoke_prompt	Invoke a function from the provided prompt
invoke_prompt_stream	Invoke a function from the provided prompt and stream the results
invoke_stream	<p>Execute one or more stream functions.</p> <p>This will execute the functions in the order they are provided, if a list of functions is provided. When multiple functions are provided only the last one is streamed, the rest is executed as a pipeline.</p> <pre>:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.kernel.str None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.kernel.str None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.kernel.dict[str, Any]> :param return_function_results: If True, the function results are yielded as a list[FunctionResult] :type return_function_results: bool :param in addition to the streaming content: :param otherwise only the streaming content is yielded.: :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.kernel.dict[str, Any]></pre>

invoke

Execute one or more functions.

When multiple functions are passed the FunctionResult of each is put into a list.

:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.kernel.str | None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.kernel.str | None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.kernel.dict[str, Any]> :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.kernel.dict[str, Any]>

Python

```
async invoke(function: KernelFunction | None = None, arguments: KernelArguments | None = None, function_name: str | None = None, plugin_name: str | None = None, metadata: dict[str, Any] = {}, **kwargs: Any) -> FunctionResult | None
```

Parameters

[] Expand table

Name	Description
function	default value: None
arguments	default value: None
function_name	default value: None
plugin_name	default value: None
metadata	default value: {}

Returns

[] Expand table

Type	Description
FunctionResult list[FunctionResult] None	The result of the function(s)

invoke_prompt

Invoke a function from the provided prompt

Python

```
async invoke_prompt(function_name: str, plugin_name: str, prompt: str,
arguments: KernelArguments | None = None, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
**kwargs: Any) -> FunctionResult | None
```

Parameters

[] Expand table

Name	Description
function_name Required*	str The name of the function
plugin_name Required*	str The name of the plugin
prompt Required*	str The prompt to use
arguments	<xref:<xref:semantic_kernel.kernel.KernelArguments None>> The arguments to pass to the function(s), optional default value: None
template_format	<xref:<xref:semantic_kernel.kernel.str None>> The format of the prompt template default value: semantic-kernel
kwargs Required*	dict[str,<xref: Any>] arguments that can be used instead of supplying KernelArguments

Returns

[] Expand table

Type	Description
FunctionResult list[FunctionResult] None	The result of the function(s)

invoke_prompt_stream

Invoke a function from the provided prompt and stream the results

Python

```
async invoke_prompt_stream(function_name: str, plugin_name: str, prompt: str, arguments: KernelArguments | None = None, template_format: Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel', return_function_results: bool | None = False, **kwargs: Any) -> AsyncIterable[list[semantic_kernel.contents.streaming_content_mixin.StreamingContentMixin] | FunctionResult | list[semantic_kernel.functions.function_result.FunctionResult]]
```

Parameters

[] Expand table

Name	Description
function_name Required*	str The name of the function
plugin_name Required*	str The name of the plugin
prompt Required*	str The prompt to use
arguments	<xref:<xref:semantic_kernel.kernel.KernelArguments None>> The arguments to pass to the function(s), optional default value: None
template_format	<xref:<xref:semantic_kernel.kernel.str None>> The format of the prompt template default value: semantic-kernel
kwargs Required*	dict[str,<xref: Any>] arguments that can be used instead of supplying KernelArguments
return_function_results	default value: False

Returns

[+] Expand table

Type	Description
<code>Asynclitable[StreamingContentMixin]</code>	The content of the stream of the last function provided.

invoke_stream

Execute one or more stream functions.

This will execute the functions in the order they are provided, if a list of functions is provided. When multiple functions are provided only the last one is streamed, the rest is executed as a pipeline.

:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.kernel.str | None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.kernel.str | None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.kernel.dict[str, Any]> :param return_function_results: If True, the function results are yielded as a list[FunctionResult] :type return_function_results: bool :param in addition to the streaming content: :param otherwise only the streaming content is yielded.: :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.kernel.dict[str, Any]>

Python

```
async invoke_stream(function: KernelFunction | None = None, arguments: KernelArguments | None = None, function_name: str | None = None, plugin_name: str | None = None, metadata: dict[str, Any] = {}, return_function_results: bool = False, **kwargs: Any) -> AsyncGenerator[list['StreamingContentMixin'] | FunctionResult | list[semantic_kernel.functions.function_result.FunctionResult], Any]
```

Parameters

[+] Expand table

Name	Description
<code>function</code>	default value: None
<code>arguments</code>	default value: None
<code>function_name</code>	default value: None
<code>plugin_name</code>	default value: None
<code>metadata</code>	default value: {}
<code>return_function_results</code>	default value: False

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [`ConfigDict`] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [`FieldInfo`][`pydantic.fields.FieldInfo`].

This replaces `Model.fields` from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_service_selector':  
    FieldInfo(annotation=AIServiceSelector, required=False,  
    default_factory=AIServiceSelector), 'auto_function_invocation_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'function_invocation_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'plugins':  
    FieldInfo(annotation=dict[str, KernelPlugin], required=False,  
    default_factory=dict), 'prompt_rendering_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'retry_mechanism':  
    FieldInfo(annotation=RetryMechanismBase, required=False,  
    default_factory=PassThroughWithoutRetry), 'services':  
    FieldInfo(annotation=dict[str, AIServiceClientBase], required=False,  
    default_factory=dict)}
```

plugins

The plugins to be used by the kernel

Python

```
plugins: dict[str, KernelPlugin]
```

services

The services to be used by the kernel

Python

```
services: dict[str, AIServiceClientBase]
```

ai_service_selector

The AI service selector to be used by the kernel

Python

```
ai_service_selector: AIServiceSelector
```

retry_mechanism

The retry mechanism to be used by the kernel

Python

```
retry_mechanism: RetryMechanismBase
```

auto_function_invocation_filters

Python

```
auto_function_invocation_filters: list[tuple[int, CALLABLE_FILTER_TYPE]]
```

function_invocation_filters

Python

```
function_invocation_filters: list[tuple[int, CALLABLE_FILTER_TYPE]]
```

prompt_rendering_filters

Python

```
prompt_rendering_filters: list[tuple[int, CALLABLE_FILTER_TYPE]]
```

kernel_pydantic Module

Reference

Classes

 [Expand table](#)

Kernel BaseModel Base class for all pydantic models in the SK.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

KernelBaseModel Class

Reference

Base class for all pydantic models in the SK.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance pydantic.main.BaseModel → KernelBaseModel

Constructor

Python

```
KernelBaseModel()
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
```

```
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {}
```

memory_query_result Module

Reference

Classes

 Expand table

[MemoryQueryResult](#)

Note: This class is experimental and may change in the future.

Initialize a new instance of MemoryQueryResult.

MemoryQueryResult Class

Reference

Note: This class is experimental and may change in the future.

Initialize a new instance of MemoryQueryResult.

Inheritance builtins.object → MemoryQueryResult

Constructor

Python

```
MemoryQueryResult(is_reference: bool, external_source_name: str | None, id: str, description: str | None, text: str | None, additional_metadata: str | None, embedding: ndarray | None, relevance: float)
```

Parameters

[] Expand table

Name	Description
record. Required*	<xref:embedding {ndarray} -- The embedding> of <xref:the>
source. Required*	<xref:external_source_name {Optional}> [str]<xref:> -- The name> of <xref:the external>
record. Required*	
record. Required*	
record. Required*	
record. Required*	
query. Required*	<xref:><xref:relevance {float} -- The relevance> of <xref:the record to a known>>
is_reference	

Name	Description
Required*	
external_source_name	
Required*	
id	
Required*	
description	
Required*	
text	
Required*	
additional_metadata	
Required*	
embedding	
Required*	
relevance	
Required*	

Methods

[] [Expand table](#)

[from_memory_record](#) Create a new instance of MemoryQueryResult from a MemoryRecord.

from_memory_record

Create a new instance of MemoryQueryResult from a MemoryRecord.

Python

```
static from_memory_record(record: MemoryRecord, relevance: float) ->
MemoryQueryResult
```

Parameters

[] [Expand table](#)

Name	Description
from. Required*	<xref:<xref:record {MemoryRecord} -- The MemoryRecord to create the MemoryQueryResult>>
query. Required*	<xref:<xref:relevance {float} -- The relevance> of <xref:the record to a known>>
record Required*	
relevance Required*	

Returns

[+] Expand table

Type	Description
	MemoryQueryResult – The created MemoryQueryResult.

Attributes

additional_metadata

Python

```
additional_metadata: str | None
```

description

Python

```
description: str | None
```

embedding

Python

```
embedding: ndarray | None
```

external_source_name

Python

```
external_source_name: str | None
```

id

Python

```
id: str
```

is_experimental

Python

```
is_experimental = True
```

is_reference

Python

```
is_reference: bool
```

relevance

Python

```
relevance: float
```

text

Python

```
text: str | None
```

memory_record Module

Reference

Classes

 Expand table

[MemoryRecord](#)

Note: This class is experimental and may change in the future.

Initialize a new instance of MemoryRecord.

MemoryRecord Class

Reference

Note: This class is experimental and may change in the future.

Initialize a new instance of MemoryRecord.

Inheritance builtins.object → MemoryRecord

Constructor

Python

```
MemoryRecord(is_reference: bool, external_source_name: str | None, id: str, description: str | None, text: str | None, additional_metadata: str | None, embedding: ndarray | None, key: str | None = None, timestamp: datetime | None = None)
```

Parameters

 Expand table

Name	Description
is_reference Required*	
external_source_name Required*	
id Required*	
description Required*	
text Required*	
additional_metadata Required*	
embedding Required*	
key	default value: None
timestamp	default value: None

Methods

[Expand table](#)

local_record	Create a local record.
reference_record	Create a reference record.

local_record

Create a local record.

Python

```
static local_record(id: str, text: str, description: str | None,
additional_metadata: str | None, embedding: ndarray, timestamp: datetime
| None = None) -> MemoryRecord
```

Parameters

[\[\]](#) Expand table

Name	Description
record. Required*	<xref:timestamp {Optional}>[datetime]<xref:> -- The timestamp> of <xref:the>
record. Required*	
id Required*	
text Required*	
description Required*	
additional_metadata Required*	
embedding Required*	
timestamp	default value: None

Returns

[\[\]](#) Expand table

Type	Description
	MemoryRecord – The local record.

reference_record

Create a reference record.

Python

```
static reference_record(external_id: str, source_name: str, description: str | None, additional_metadata: str | None, embedding: ndarray) -> MemoryRecord
```

Parameters

[+] Expand table

Name	Description
record. Required*	<xref:embedding {ndarray} -- The embedding> of <xref:the>
source. Required*	<xref:<xref:source_name {str} -- The name> of <xref:the external>>
record. Required*	
record. Required*	
record. Required*	
external_id Required*	
source_name Required*	
description Required*	
additional_metadata Required*	
embedding Required*	

Returns

[+] Expand table

Type	Description
	MemoryRecord – The reference record.

Attributes

additional_metadata

description

embedding

id

text

timestamp

is_experimental

Python

```
is_experimental = True
```

memory_store_base Module

Reference

Classes

 Expand table

[MemoryStoreBase](#)

Note: This class is experimental and may change in the future.

MemoryStoreBase Class

Reference

Note: This class is experimental and may change in the future.

Inheritance [ABC](#) → MemoryStoreBase

Constructor

Python

```
MemoryStoreBase()
```

Methods

[\[+\] Expand table](#)

close	Async close connection, invoked by <code>MemoryStoreBase.aexit()</code>
create_collection	Creates a new collection in the data store.
delete_collection	Deletes a collection from the data store.
does_collection_exist	Determines if a collection exists in the data store.
get	Gets a memory record from the data store. Does not guarantee that the collection exists.
get_batch	Gets a batch of memory records from the data store. Does not guarantee that the collection exists.
get_collections	Gets all collection names in the data store.
get_nearest_match	Gets the nearest match to an embedding of type float. Does not guarantee that the collection exists.
get_nearest_matches	Gets the nearest matches to an embedding of type float. Does not guarantee that the collection exists.
remove	Removes a memory record from the data store. Does not guarantee that the collection exists.
remove_batch	Removes a batch of memory records from the data store. Does not guarantee that the collection exists.

<code>upsert</code>	Upserts a memory record into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.
<code>upsert_batch</code>	Upserts a group of memory records into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

close

Async close connection, invoked by `MemoryStoreBase.aexit()`

Python

```
async close()
```

create_collection

Creates a new collection in the data store.

Python

```
abstract async create_collection(collection_name: str) -> None
```

Parameters

[+] Expand table

Name	Description
<code>embeddings.</code> Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
<code>collection_name</code> Required*	

Returns

[+] Expand table

Type	Description
	None

delete_collection

Deletes a collection from the data store.

Python

```
abstract async delete_collection(collection_name: str) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

does_collection_exist

Determines if a collection exists in the data store.

Python

```
abstract async does_collection_exist(collection_name: str) -> bool
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
collection_name Required*	

Returns

[\[\] Expand table](#)

Type	Description
	bool – True if given collection exists, False if not.

get

Gets a memory record from the data store. Does not guarantee that the collection exists.

Python

```
abstract async get(collection_name: str, key: str, with_embedding: bool)  
-> MemoryRecord
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
get. Required*	<xref:<xref:key {str} -- The unique id associated with the memory record to>>
true Required*	<xref:<xref:with_embedding {bool} -- If>>
record.	<xref:<xref:the embedding will be returned in the memory>>

Name	Description
Required*	
collection_name Required*	
key Required*	
with_embedding Required*	

Returns

[\[\] Expand table](#)

Type	Description
	MemoryRecord – The memory record if found

get_batch

Gets a batch of memory records from the data store. Does not guarantee that the collection exists.

Python

```
abstract async get_batch(collection_name: str, keys: list[str], with_embeddings: bool) -> list[semantic_kernel.memory.memory_record.MemoryRecord]
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
get. Required*	<xref:keys {List}>[str]<xref:-- The unique ids associated with the memory records to>
true	<xref:<xref:with_embeddings {bool} -- If>>

Name	Description
Required*	
records. Required*	<xref:<xref:the embedding will be returned in the memory>>
collection_name Required*	
keys Required*	
with_embeddings Required*	

Returns

[] Expand table

Type	Description
	List[MemoryRecord] – The memory records associated with the unique keys provided.

get_collections

Gets all collection names in the data store.

Python

```
abstract async get_collections() -> list[str]
```

Returns

[] Expand table

Type	Description
	List[str] – A group of collection names.

get_nearest_match

Gets the nearest match to an embedding of type float. Does not guarantee that the collection exists.

Python

```
abstract async get_nearest_match(collection_name: str, embedding:  
ndarray, min_relevance_score: float, with_embedding: bool) ->  
tuple[semantic_kernel.memory.memory_record.MemoryRecord, float]
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
with. Required*	<xref:<xref:embedding {ndarray} -- The embedding to compare the collection's embeddings>>
result. Required*	<xref:<xref:min_relevance_score {float} -- The minimum relevance threshold for returned>>
true Required*	<xref:<xref:with_embedding {bool} -- If>>
record. Required*	<xref:<xref:the embeddings will be returned in the memory>>
collection_name Required*	
embedding Required*	
min_relevance_score Required*	
with_embedding Required*	

Returns

[+] Expand table

Type	Description
	Tuple[MemoryRecord, float] – A tuple consisting of the MemoryRecord and the similarity score as a float.

get_nearest_matches

Gets the nearest matches to an embedding of type float. Does not guarantee that the collection exists.

Python

```
abstract async get_nearest_matches(collection_name: str, embedding: ndarray, limit: int, min_relevance_score: float, with_embeddings: bool) -> list[tuple[semantic_kernel.memory.MemoryRecord, float]]
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
with. Required*	<xref:<xref:embedding {ndarray} -- The embedding to compare the collection's embeddings>>
return. Required*	<xref:<xref:limit {int} -- The maximum number> of <xref:similarity results to>>
results. Required*	<xref:<xref:min_relevance_score {float} -- The minimum relevance threshold for returned>>
true Required*	<xref:<xref:with_embeddings {bool} -- If>>
records. Required*	<xref:<xref:the embeddings will be returned in the memory>>
collection_name Required*	
embedding Required*	
limit Required*	
min_relevance_score Required*	
with_embeddings	

Name	Description
Required*	

Returns

[+] Expand table

Type	Description
	List[Tuple[MemoryRecord, float]] – A list of tuples where item1 is a MemoryRecord and item2 is its similarity score as a float.

remove

Removes a memory record from the data store. Does not guarantee that the collection exists.

Python

```
abstract async remove(collection_name: str, key: str) -> None
```

Parameters

[+] Expand table

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
remove. Required*	<xref:<xref:key {str} -- The unique id associated with the memory record to>>
collection_name Required*	
key Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

remove_batch

Removes a batch of memory records from the data store. Does not guarantee that the collection exists.

Python

```
abstract async remove_batch(collection_name: str, keys: list[str]) -> None
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
remove. Required*	<xref:keys {List}>[str]<xref:-- The unique ids associated with the memory records to>
collection_name Required*	
keys Required*	

Returns

[\[\] Expand table](#)

Type	Description
	None

upsert

Upserts a memory record into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

Python

```
abstract async upsert(collection_name: str, record: MemoryRecord) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
upsert. Required*	<xref:<xref:record {MemoryRecord} -- The memory record to>>
collection_name Required*	
record Required*	

Returns

[\[\] Expand table](#)

Type	Description
	str – The unique identifier for the memory record.

upsert_batch

Upserts a group of memory records into the data store. Does not guarantee that the collection exists. If the record already exists, it will be updated. If the record does not exist, it will be created.

Python

```
abstract async upsert_batch(collection_name: str, records:
```

```
list[semantic_kernel.memory.memory_record.MemoryRecord]) -> list[str]
```

Parameters

[\[\] Expand table](#)

Name	Description
embeddings. Required*	<xref:<xref:collection_name {str} -- The name associated with a collection of>>
upsert. Required*	<xref:<xref:records {MemoryRecord} -- The memory records to>>
collection_name Required*	
records Required*	

Returns

[\[\] Expand table](#)

Type	Description
	List[str] – The unique identifiers for the memory records.

Attributes

is_experimental

Python

```
is_experimental = True
```

kernel_reliability_extension Module

Reference

Classes

[+] Expand table

<p><code>KernelReliabilityExtension</code></p>	Create a new model by parsing and validating input data from keyword arguments.
--	---

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

KernelReliabilityExtension Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `KernelBaseModel` → `KernelReliabilityExtension`

`ABC` → `KernelReliabilityExtension`

Constructor

Python

```
KernelReliabilityExtension(*, retry_mechanism: RetryMechanismBase = None)
```

Keyword-Only Parameters

[+] Expand table

Name	Description
<code>retry_mechanism</code> Required*	

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'retry_mechanism':  
FieldInfo(annotation=RetryMechanismBase, required=False,  
default_factory=PassThroughWithoutRetry)}
```

retry_mechanism

Python

```
retry_mechanism: RetryMechanismBase
```

pass_through_without_retry Module

Reference

Classes

[+] Expand table

`PassThroughWithoutRetry` A retry mechanism that does not retry.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

PassThroughWithoutRetry Class

Reference

A retry mechanism that does not retry.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance `RetryMechanismBase` → `PassThroughWithoutRetry`
`KernelBaseModel` → `PassThroughWithoutRetry`

Constructor

Python

```
PassThroughWithoutRetry()
```

Methods

[+] Expand table

<code>execute_with_retry</code>	Executes the given action with retry logic.
---------------------------------	---

`execute_with_retry`

Executes the given action with retry logic.

Python

```
async execute_with_retry(action: Callable[[], Awaitable[T]]) ->
Awaitable[T]
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>{Callable[]}</code> Required*	<code>[] (<xref:action>)</code>
<code>exception.</code> Required*	<code>Awaitable[<xref:T>]]<xref:></code> -- The action to retry on>
<code>action</code> Required*	

Returns

[\[\] Expand table](#)

Type	Description
	<code>Awaitable[T]</code> – An awaitable that will return the result of the action.

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding `ComputedFieldInfo` objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to `[ConfigDict]` [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {}
```

retry_mechanism_base Module

Reference

Classes

 Expand table

RetryMechanismBase

RetryMechanismBase Class

Reference

Inheritance [ABC](#) → RetryMechanismBase

Constructor

Python

```
RetryMechanismBase()
```

Methods

[\[+\] Expand table](#)

execute_with_retry	Executes the given action with retry logic.
------------------------------------	---

execute_with_retry

Executes the given action with retry logic.

Python

```
abstract async execute_with_retry(action: Callable[[], Awaitable[T]]) -> Awaitable[T]
```

Parameters

[\[+\] Expand table](#)

Name	Description
<code>{Callable[]}</code> Required*	<code>[]</code> (<xref:action>
<code>exception.</code> Required*	<code>Awaitable[<xref:T>]]</code> <xref:> -- The action to retry on>
<code>action</code> Required*	

Returns

[\[\] Expand table](#)

Type	Description
	Awaitable[T] – An awaitable that will return the result of the action.

kernel_json_schema_builder Module

Reference

Classes

[+] Expand table

KernelJsonSchemaBuilder

KernelJsonSchemaBuilder Class

Reference

Inheritance builtins.object → KernelJsonSchemaBuilder

Constructor

Python

```
KernelJsonSchemaBuilder()
```

Methods

[+] Expand table

build	Builds JSON schema for a given parameter type.
build_from_type_name	Builds JSON schema for a given parameter type name.
build_model_schema	Builds JSON schema for a given model.
get_json_schema	Gets JSON schema for a given parameter type.

build

Builds JSON schema for a given parameter type.

Python

```
build(parameter_type: type | str, description: str | None = None) ->
dict[str, Any]
```

Parameters

[+] Expand table

Name	Description
parameter_type Required*	

Name	Description
description	default value: None

build_from_type_name

Builds JSON schema for a given parameter type name.

Python

```
build_from_type_name(parameter_type: str, description: str | None = None) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
parameter_type Required*	
description	default value: None

build_model_schema

Builds JSON schema for a given model.

Python

```
build_model_schema(model: type, description: str | None = None) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
model Required*	

Name	Description
description	default value: None

get_json_schema

Gets JSON schema for a given parameter type.

Python

```
get_json_schema(parameter_type: type) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
parameter_type Required*	

block Module

Reference

Classes

[\[\] Expand table](#)

Block Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Block Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [KernelBaseModel](#) → Block

Constructor

Python

```
Block(*, content: str)
```

Keyword-Only Parameters

[] [Expand table](#)

Name	Description
content Required*	

Methods

[] [Expand table](#)

```
content_strip
```

content_strip

Python

```
content_strip(content: str)
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':  
FieldInfo(annotation=str, required=True)}
```

content

```
Python
```

```
content: str
```

type

```
Python
```

```
type: ClassVar[BlockTypes] = 1
```

block_types Module

Reference

Enums

[\[\] Expand table](#)

BlockTypes

BlockTypes Enum

Reference

Inheritance [Enum](#) → BlockTypes

Constructor

Python

```
BlockTypes(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[+] [Expand table](#)

CODE
FUNCTION_ID
NAMED_ARG
TEXT
UNDEFINED
VALUE
VARIABLE

code_block Module

Reference

Classes

[] [Expand table](#)

CodeBlock Create a code block.

A code block is a block that usually contains functions to be executed by the kernel. It consists of a list of tokens that can be either a function_id, value, a variable or a named argument.

If the first token is not a function_id but a variable or value, the rest of the tokens will be ignored. Only the first argument for the function can be a variable or value, the rest of the tokens have be named arguments.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

CodeBlock Class

Reference

Create a code block.

A code block is a block that usually contains functions to be executed by the kernel. It consists of a list of tokens that can be either a function_id, value, a variable or a named argument.

If the first token is not a function_id but a variable or value, the rest of the tokens will be ignored. Only the first argument for the function can be a variable or value, the rest of the tokens have be named arguments.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [Block](#) → CodeBlock

Constructor

Python

```
CodeBlock(*, content: str, tokens:  
list[semantic_kernel.template_engine.blocks.block.Block] = None)
```

Parameters

[Expand table](#)

Name	Description
content Required*	The content of the code block.
t tokens Required*	The list of tokens that compose the code block, if empty, will be created by the CodeTokenizer.

Keyword-Only Parameters

[+] Expand table

Name	Description
content Required*	
tokens Required*	

Methods

[+] Expand table

<code>check_tokens</code>	Check the tokens in the list. If the first token is a value or variable, the rest of the tokens will be ignored. If the first token is a function_id, then the next token can be a value, variable or named_arg, the rest have to be named_args.
<code>parse_content</code>	Parse the content of the code block and tokenize it. If tokens are already present, skip the tokenizing.
<code>render_code</code>	Render the code block. If the first token is a function_id, it will call the function from the plugin collection. Otherwise it is a value or variable and those are then rendered directly.

check_tokens

Check the tokens in the list.

If the first token is a value or variable, the rest of the tokens will be ignored. If the first token is a function_id, then the next token can be a value, variable or named_arg, the rest have to be named_args.

Python

```
check_tokens:  
list[semantic_kernel.template_engine.blocks.block.Block]) ->  
list[semantic_kernel.template_engine.blocks.block.Block]
```

Parameters

[+] Expand table

Name	Description
tokens Required*	

Exceptions

[+] Expand table

Type	Description
CodeBlockTokenError	If the content does not contain at least one token.
CodeBlockTokenError	If the first token is a named argument.
CodeBlockTokenError	If the second token is not a value or variable.
CodeBlockTokenError	If a token is not a named argument after the second token.

parse_content

Parse the content of the code block and tokenize it.

If tokens are already present, skip the tokenizing.

Python

```
parse_content(fields: Any) -> Any
```

Parameters

[+] Expand table

Name	Description
fields Required*	

Exceptions

[\[\] Expand table](#)

Type	Description
CodeBlockTokenError	If the content does not contain at least one token.
CodeBlockTokenError	If the first token is a named argument.
CodeBlockTokenError	If the second token is not a value or variable.
CodeBlockTokenError	If a token is not a named argument after the second token.

render_code

Render the code block.

If the first token is a function_id, it will call the function from the plugin collection. Otherwise it is a value or variable and those are then rendered directly.

Python

```
async render_code(kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	
arguments Required*	

Exceptions

[\[\] Expand table](#)

Type	Description
CodeBlockTokenError	If the content does not contain at least one token.
CodeBlockTokenError	If the first token is a named argument.

Type	Description
CodeBlockTokenError	If the second token is not a value or variable.
CodeBlockTokenError	If a token is not a named argument after the second token.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [[pydantic.config.ConfigDict](#)].

Python

```
model_config: ClassVar[ConfigDict] = { 'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][[pydantic.fields.FieldInfo](#)].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = { 'content':
FieldInfo(annotation=str, required=True), 'tokens':
FieldInfo(annotation=list[Block], required=False, default_factory=list)}
```

tokens

Python

```
tokens: list[semantic_kernel.template_engine.blocks.block.Block]
```

type

Python

```
type: ClassVar[BlockTypes] = 3
```

function_id_block Module

Reference

Classes

 [Expand table](#)

FunctionIdBlock Block to represent a function id. It can be used to call a function from a plugin.

The content is parsed using a regex, that returns either a plugin and function name or just a function name, depending on the content.

Anything other then that and a ValueError is raised.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

FunctionIdBlock Class

Reference

Block to represent a function id. It can be used to call a function from a plugin.

The content is parsed using a regex, that returns either a plugin and function name or just a function name, depending on the content.

Anything other then that and a ValueError is raised.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [Block](#) → FunctionIdBlock

Constructor

Python

```
FunctionIdBlock(*, content: str, function_name: str | None = '',
plugin_name: str | None = None)
```

Parameters

[\[\]](#) Expand table

Name	Description
content Required*	<code>str</code> The content of the block.
function_name Required*	<code><xref:Optional>[str],<xref: optional></code> The function name.
plugin_name Required*	<code><xref:Optional>[str],<xref: optional></code> The plugin name.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
content Required*	
function_name Required*	
plugin_name Required*	

Methods

[\[\] Expand table](#)

parse_content	Parse the content of the function id block and extract the plugin and function name. If both are present in the fields, return the fields as is. Otherwise use the regex to extract the plugin and function name.
render	

parse_content

Parse the content of the function id block and extract the plugin and function name.

If both are present in the fields, return the fields as is. Otherwise use the regex to extract the plugin and function name.

Python

```
parse_content(fields: dict[str, Any]) -> dict[str, Any]
```

Parameters

[\[\] Expand table](#)

Name	Description
fields Required*	

render

Python

```
render(*_: tuple['Kernel',  
typing.Optional[ForwardRef('KernelArguments')]]) -> str
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':  
FieldInfo(annotation=str, required=True), 'function_name':  
FieldInfo(annotation=Union[str, NoneType], required=False, default=''),
```

```
'plugin_name': FieldInfo(annotation=Union[str, NoneType], required=False,  
default=None)}
```

function_name

Python

```
function_name: str | None
```

plugin_name

Python

```
plugin_name: str | None
```

type

Python

```
type: ClassVar[BlockTypes] = 6
```

named_arg_block Module

Reference

Classes

[] [Expand table](#)

[NamedArgBlock](#) Create a named argument block.

A named arg block is used to add arguments to a function call. It needs to be combined with a function_id block to be useful. Inside a code block, if the first block is a function_id block, the first block can be a variable or value block, anything else must be a named arg block.

The value inside the NamedArgBlock can be a ValBlock or a VarBlock.

Create a new model by parsing and validating input data from keyword arguments.

Raises [*ValidationError*][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

NamedArgBlock Class

Reference

Create a named argument block.

A named arg block is used to add arguments to a function call. It needs to be combined with a function_id block to be useful. Inside a code block, if the first block is a function_id block, the first block can be a variable or value block, anything else must be a named arg block.

The value inside the NamedArgBlock can be a ValBlock or a VarBlock.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [Block](#) → NamedArgBlock

Constructor

Python

```
NamedArgBlock(*, content: str, name: str | None = None, value: ValBlock |  
None = None, variable: VarBlock | None = None)
```

Parameters

[\[+\] Expand table](#)

Name	Description
str Required*	<xref:<xref:name ->> The content of the named argument block, the name and value separated by an equal sign, for instance arg1=\$var.
str Required*	The name of the argument.
ValBlock Required*	<xref:<xref:value ->> The value of the argument.

Name	Description
VarBlock Required*	<xref:<xref:variable ->> The variable of the argument. Either the value or variable field is used.

Keyword-Only Parameters

[] [Expand table](#)

Name	Description
content Required*	
name Required*	
value Required*	
variable Required*	

Examples

```
{{ plugin.function arg1=$var }} {{ plugin.function arg1='value' }} {{ plugin.function 'value'
arg2=$var }} {{ plugin.function $var arg2='value' }} {{ plugin_function arg1=$var1
arg2=$var2 arg3='value' }}
```

Methods

[] [Expand table](#)

parse_content	Parse the content of the named argument block and extract the name and value. If the name and either value or variable is present the parsing is skipped. Otherwise the content is parsed using a regex to extract the name and value. Those are then turned into Blocks.
render	

parse_content

Parse the content of the named argument block and extract the name and value.

If the name and either value or variable is present the parsing is skipped. Otherwise the content is parsed using a regex to extract the name and value. Those are then turned into Blocks.

Python

```
parse_content(fields: Any) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
fields Required*	

Exceptions

[\[\] Expand table](#)

Type	Description
NamedArgBlockSyntaxError	If the content does not match the named argument syntax.

render

Python

```
render(kernel: Kernel, arguments: KernelArguments | None = None) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
kernel Required*	
arguments	default value: None

Exceptions

[+] Expand table

Type	Description
NamedArgBlockSyntaxError	If the content does not match the named argument syntax.

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':
FieldInfo(annotation=str, required=True), 'name':
FieldInfo(annotation=Union[str, NoneType], required=False, default=None),
'value': FieldInfo(annotation=Union[ValBlock, NoneType], required=False,
```

```
default=None), 'variable': FieldInfo(annotation=Union[VarBlock,  
NoneType], required=False, default=None)}
```

name

Python

```
name: str | None
```

type

Python

```
type: ClassVar[BlockTypes] = 7
```

value

Python

```
value: ValBlock | None
```

variable

Python

```
variable: VarBlock | None
```

symbols Module

Reference

Enums

[\[\] Expand table](#)

Symbols

Symbols Enum

Reference

Inheritance builtins.str → Symbols
Enum → Symbols

Constructor

Python

```
Symbols(value, names=None, *, module=None, qualname=None, type=None,  
start=1, boundary=None)
```

Fields

[] Expand table

BLOCK_ENDER
BLOCK_STARTER
CARRIAGE_RETURN
DBL_QUOTE
ESCAPE_CHAR
NAMED_ARG_BLOCK_SEPARATOR
NEW_LINE
SGL_QUOTE
SPACE
TAB
VAR_PREFIX

text_block Module

Reference

Classes

 Expand table

`TextBlock` Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

TextBlock Class

Reference

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Inheritance [Block](#) → TextBlock

Constructor

Python

```
TextBlock(*, content: str)
```

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

Methods

[\[\] Expand table](#)

content_strip
from_text
render

content_strip

Python

```
content_strip(content: str)
```

Parameters

[\[\] Expand table](#)

Name	Description
content Required*	

from_text

Python

```
from_text(text: str | None = None, start_index: int | None = None,  
stop_index: int | None = None)
```

Parameters

[\[\] Expand table](#)

Name	Description
text	default value: None
start_index	default value: None
stop_index	default value: None

render

Python

```
render(*_: tuple[typing.Optional[ForwardRef('Kernel')],  
typing.Optional[ForwardRef('KernelArguments')]]) -> str
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':  
FieldInfo(annotation=str, required=True)}
```

type

Python

```
type: ClassVar[BlockTypes] = 2
```

val_block Module

Reference

Classes

[\[\] Expand table](#)

ValBlock Create a value block.

A value block is used to represent a value in a template. It can be used to represent any characters. It needs to start and end with the same quote character, can be both single or double quotes, as long as they are not mixed.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

ValBlock Class

Reference

Create a value block.

A value block is used to represent a value in a template. It can be used to represent any characters. It needs to start and end with the same quote character, can be both single or double quotes, as long as they are not mixed.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [Block](#) → ValBlock

Constructor

Python

```
ValBlock(*, content: str, value: str | None = '', quote: str | None = "")
```

Parameters

[\[+\] Expand table](#)

Name	Description
str Required*	<xref:<xref:quote ->> The content of the value block.
str Required*	The value of the block.
str Required*	The quote used to wrap the value.

Keyword-Only Parameters

[\[\] Expand table](#)

Name	Description
content Required*	
value Required*	
quote	default value: '

Examples

'value' "value" 'value with "quotes'" "value with 'quotes'"

Methods

[\[\] Expand table](#)

[parse_content](#) Parse the content and extract the value and quote.

The parsing is based on a regex that returns the value and quote. if the 'value' is already present then the parsing is skipped.

[render](#)

parse_content

Parse the content and extract the value and quote.

The parsing is based on a regex that returns the value and quote. if the 'value' is already present then the parsing is skipped.

Python

```
parse_content(fields: Any) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
fields Required*	

render

Python

```
render(*_: tuple['Kernel',  
typing.Optional[ForwardRef('KernelArguments')]]) -> str
```

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*] [`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':  
    FieldInfo(annotation=str, required=True), 'quote':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default=''),  
    'value': FieldInfo(annotation=Union[str, NoneType], required=False,  
    default='')}
```

quote

Python

```
quote: str | None
```

type

Python

```
type: ClassVar[BlockTypes] = 5
```

value

Python

```
value: str | None
```

var_block Module

Reference

Classes

[\[\] Expand table](#)

`VarBlock` Create a variable block.

A variable block is used to add a variable to a template. It get's rendered from KernelArguments, if the variable is not found a warning is logged and an empty string is returned. The variable must start with \$ and be followed by a valid variable name. A valid variable name is a string of letters, numbers and underscores.

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

VarBlock Class

Reference

Create a variable block.

A variable block is used to add a variable to a template. It gets rendered from KernelArguments, if the variable is not found a warning is logged and an empty string is returned. The variable must start with \$ and be followed by a valid variable name. A valid variable name is a string of letters, numbers and underscores.

Create a new model by parsing and validating input data from keyword arguments.

Raises [ValidationError][pydantic_core.ValidationError] if the input data cannot be validated to form a valid model.

self is explicitly positional-only to allow *self* as a field name.

Inheritance [Block](#) → VarBlock

Constructor

Python

```
VarBlock(*, content: str, name: str | None = '')
```

Parameters

[+] Expand table

Name	Description
str Required*	<xref:<xref:name ->> The content of the variable block, the name of the variable.
str Required*	The name of the variable.

Keyword-Only Parameters

[+] Expand table

Name	Description
content Required*	
name Required*	

Examples

```
$var $test_var
```

Methods

[\[\] Expand table](#)

parse_content Parse the content and extract the name.

The parsing is based on a regex that returns the name. if the 'name' is already present then the parsing is skipped.

render Render the variable block with the given arguments. If the variable is not found in the arguments, return an empty string.

parse_content

Parse the content and extract the name.

The parsing is based on a regex that returns the name. if the 'name' is already present then the parsing is skipped.

Python

```
parse_content(fields: Any) -> Any
```

Parameters

[\[\] Expand table](#)

Name	Description
fields Required*	

render

Render the variable block with the given arguments. If the variable is not found in the arguments, return an empty string.

Python

```
render(_: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[+] Expand table

Name	Description
_ Required*	
arguments	default value: None

Attributes

model_computed_fields

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

model_config

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,  
'populate_by_name': True, 'validate_assignment': True}
```

model_fields

Metadata about the fields defined on the model, mapping of field names to [FieldInfo][pydantic.fields.FieldInfo].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'content':  
    FieldInfo(annotation=str, required=True), 'name':  
    FieldInfo(annotation=Union[str, NoneType], required=False, default='')}
```

name

Python

```
name: str | None
```

type

Python

```
type: ClassVar[BlockTypes] = 4
```

code_tokenizer Module

Reference

Classes

[] [Expand table](#)

CodeTokenizer

CodeTokenizer Class

Reference

Inheritance builtins.object → CodeTokenizer

Constructor

Python

```
CodeTokenizer()
```

Methods

[+] Expand table

tokenize

Python

```
static tokenize(text: str) ->
list[semantic_kernel.template_engine.blocks.block.Block]
```

Parameters

[+] Expand table

Name	Description
text Required*	

code_renderer Module

Reference

Classes

 [Expand table](#)

CodeRenderer	Protocol for dynamic code blocks that need async IO to be rendered.
------------------------------	---

CodeRenderer Class

Reference

Protocol for dynamic code blocks that need async IO to be rendered.

Inheritance [Protocol](#) → CodeRenderer

Constructor

Python

```
CodeRenderer(*args, **kwargs)
```

Methods

[\[\]](#) [Expand table](#)

render_code	Render the block using the given context.
-----------------------------	---

render_code

Render the block using the given context.

Python

```
async render_code(kernel: Kernel, arguments: KernelArguments) -> str
```

Parameters

[\[\]](#) [Expand table](#)

Name	Description
context Required*	kernel execution context
kernel Required*	
arguments	

Name	Description
Required*	

Returns

[\[\] Expand table](#)

Type	Description
	Rendered content

text_renderer Module

Reference

Classes

[\[+\] Expand table](#)

TextRenderer	Protocol for static (text) blocks that don't need async rendering.
------------------------------	--

TextRenderer Class

Reference

Protocol for static (text) blocks that don't need async rendering.

Inheritance [Protocol](#) → TextRenderer

Constructor

Python

```
TextRenderer(*args, **kwargs)
```

Methods

[\[\] Expand table](#)

<code>render</code>	Render the block using only the given variables.
---------------------	--

render

Render the block using only the given variables.

Python

```
render(kernel: Kernel, arguments: KernelArguments | None = None) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
<code>variables</code> Required*	Optional variables used to render the block
<code>kernel</code> Required*	

Name	Description
arguments	default value: None

Returns

 [Expand table](#)

Type	Description
	Rendered content

template_tokenizer Module

Reference

Classes

 [Expand table](#)

TemplateTokenizer

TemplateTokenizer Class

Reference

Inheritance builtins.object → TemplateTokenizer

Constructor

Python

```
TemplateTokenizer()
```

Methods

[+] Expand table

tokenize

Python

```
static tokenize(text: str) ->
list[semantic_kernel.template_engine.blocks.block.Block]
```

Parameters

[+] Expand table

Name	Description
text Required*	

chat Module

Reference

Functions

store_results

Stores specific results in the context and chat prompt.

Python

```
store_results(chat_history: ChatHistory, results:  
list['ChatMessageContent'])
```

Parameters

[\[\] Expand table](#)

Name	Description
chat_history Required*	
results Required*	

experimental_decorator Module

Reference

Functions

experimental_class

```
experimental_class(cls: type) -> type
```

Parameters

[\[+\] Expand table](#)

Name	Description
cls Required*	

experimental_function

```
experimental_function(func: Callable) -> Callable
```

Parameters

[\[+\] Expand table](#)

Name	Description
func Required*	

logging Module

Reference

Functions

setup_logging

```
setup_logging()
```

naming Module

Reference

Functions

generate_random_ascii_name

Generate a series of random ASCII characters of the specified length. As example, plugin/function names can contain upper/lowercase letters, and underscores

Python

```
generate_random_ascii_name(length: int = 16) -> str
```

Parameters

[\[\] Expand table](#)

Name	Description
length	int The length of the string to generate. default value: 16

Returns

[\[\] Expand table](#)

Type	Description
	A string of random ASCII characters of the specified length.

validation Module

Reference

Kernel Class

Reference

The Kernel class is the main entry point for the Semantic Kernel. It provides the ability to run semantic/native functions, and manage plugins, memory, and AI services.

Initialize a new instance of the Kernel class.

Inheritance [KernelFilterExtension](#) → Kernel
[KernelFunctionExtension](#) → Kernel
[KernelServicesExtension](#) → Kernel
[KernelReliabilityExtension](#) → Kernel

Constructor

Python

```
Kernel(plugins: KernelPlugin | dict[str,  
semantic_kernel.functions.kernel_plugin.KernelPlugin] |  
list[semantic_kernel.functions.kernel_plugin.KernelPlugin] | None = None,  
services: AI_SERVICE_CLIENT_TYPE | list[AI_SERVICE_CLIENT_TYPE] | dict[str,  
AI_SERVICE_CLIENT_TYPE] | None = None, ai_service_selector:  
AIServiceSelector | None = None, *, retry_mechanism: RetryMechanismBase =  
None, function_invocation_filters: list[tuple[int,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE,  
collections.abc.Callable[[FILTER_CONTEXT_TYPE], None\]\], None\\]\\]\\] = None,  
prompt\\_rendering\\_filters: list\\[tuple\\[int,  
collections.abc.Callable\\[\\[FILTER\\_CONTEXT\\_TYPE,  
collections.abc.Callable\\[\\[FILTER\\_CONTEXT\\_TYPE\\], None\\\]\\\], None\\\\]\\\\]\\\\] = None,  
auto\\\\_function\\\\_invocation\\\\_filters: list\\\\[tuple\\\\[int,  
collections.abc.Callable\\\\[\\\\[FILTER\\\\_CONTEXT\\\\_TYPE,  
collections.abc.Callable\\\\[\\\\[FILTER\\\\_CONTEXT\\\\_TYPE\\\\], None\\\\\]\\\\\], None\\\\\\]\\\\\\]\\\\\\] = None\\\\\\)
```

Parameters

[] [Expand table](#)

Name	Description
plugins	<xref:KernelPlugin dict> [str ,<xref: KernelPlugin>]<xref:> list>[<xref:KernelPlugin>]<xref:> None The plugins to be used by the kernel, will be rewritten to a dict with plugin name as key

Name	Description
	default value: None
dict[str Required*	str (<xref:services> (AIServiceClientBase list[AIServiceClientBase]) The services to be used by the kernel, will be rewritten to a dict with service_id as key
None Required*	AIServiceClientBase] The services to be used by the kernel, will be rewritten to a dict with service_id as key
ai_service_selector	<xref:<xref:semantic_kernel.AIServiceSelector None>> The AI service selector to be used by the kernel, default is based on order of execution settings. default value: None
**kwargs Required*	Any Additional fields to be passed to the Kernel model, these are limited to retry_mechanism and function_invoking_handlers and function_invoked_handlers, the best way to add function_invoking_handlers and function_invoked_handlers is to use the add_function_invoking_handler and add_function_invoked_handler methods.
services	default value: None

Keyword-Only Parameters

[+] [Expand table](#)

Name	Description
retry_mechanism Required*	
function_invocation_filters Required*	
prompt_rendering_filters Required*	
auto_function_invocation_filters Required*	

Methods

invoke	<p>Execute one or more functions.</p> <p>When multiple functions are passed the FunctionResult of each is put into a list.</p> <pre>:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.str None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.str None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.dict[str, Any]> :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.dict[str, Any]></pre>
invoke_prompt	Invoke a function from the provided prompt
invoke_prompt_stream	Invoke a function from the provided prompt and stream the results
invoke_stream	<p>Execute one or more stream functions.</p> <p>This will execute the functions in the order they are provided, if a list of functions is provided. When multiple functions are provided only the last one is streamed, the rest is executed as a pipeline.</p> <pre>:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.str None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.str None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.dict[str, Any]> :param return_function_results: If True, the function results are yielded as a list[FunctionResult] :type return_function_results: bool :param in addition to the streaming content: :param otherwise only the streaming content is yielded.: :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.dict[str, Any]></pre>

invoke

Execute one or more functions.

When multiple functions are passed the FunctionResult of each is put into a list.

:param :param if this is none: :param function_name and plugin_name are used and cannot be None.:param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.str | None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.str | None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.dict[str, Any]> :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.dict[str, Any]>

Python

```
async invoke(function: KernelFunction | None = None, arguments: KernelArguments | None = None, function_name: str | None = None, plugin_name: str | None = None, metadata: dict[str, Any] = {}, **kwargs: Any) -> FunctionResult | None
```

Parameters

[] Expand table

Name	Description
function	default value: None
arguments	default value: None
function_name	default value: None
plugin_name	default value: None
metadata	default value: {}

Returns

[] Expand table

Type	Description
FunctionResult list[FunctionResult] None	The result of the function(s)

invoke_prompt

Invoke a function from the provided prompt

Python

```
async invoke_prompt(function_name: str, plugin_name: str, prompt: str,
arguments: KernelArguments | None = None, template_format:
Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel',
**kwargs: Any) -> FunctionResult | None
```

Parameters

[+] Expand table

Name	Description
function_name Required*	str The name of the function
plugin_name Required*	str The name of the plugin
prompt Required*	str The prompt to use
arguments	<xref:<xref:semantic_kernel.KernelArguments None>> The arguments to pass to the function(s), optional default value: None
template_format	<xref:<xref:semantic_kernel.str None>> The format of the prompt template default value: semantic-kernel
kwargs Required*	dict[str,<xref: Any>] arguments that can be used instead of supplying KernelArguments

Returns

[+] Expand table

Type	Description
FunctionResult list[FunctionResult] None	The result of the function(s)

invoke_prompt_stream

Invoke a function from the provided prompt and stream the results

Python

```
async invoke_prompt_stream(function_name: str, plugin_name: str, prompt: str, arguments: KernelArguments | None = None, template_format: Literal['semantic-kernel', 'handlebars', 'jinja2'] = 'semantic-kernel', return_function_results: bool | None = False, **kwargs: Any) -> AsyncIterable[list[semantic_kernel.contents.streaming_content_mixin.StreamingContentMixin] | FunctionResult | list[semantic_kernel.functions.function_result.FunctionResult]]
```

Parameters

[] Expand table

Name	Description
function_name Required*	str The name of the function
plugin_name Required*	str The name of the plugin
prompt Required*	str The prompt to use
arguments	<xref:<xref:semantic_kernel.KernelArguments None>> The arguments to pass to the function(s), optional default value: None
template_format	<xref:<xref:semantic_kernel.str None>> The format of the prompt template default value: semantic-kernel
kwargs Required*	dict[str,<xref: Any>] arguments that can be used instead of supplying KernelArguments
return_function_results	default value: False

Returns

[] Expand table

Type	Description
<code>AsyncIterable[StreamingContentMixin]</code>	The content of the stream of the last function provided.

invoke_stream

Execute one or more stream functions.

This will execute the functions in the order they are provided, if a list of functions is provided. When multiple functions are provided only the last one is streamed, the rest is executed as a pipeline.

:param :param if this is none: :param function_name and plugin_name are used and cannot be None.: :param arguments: The arguments to pass to the function(s), optional :type arguments: <xref:semantic_kernel.KernelArguments> :param function_name: The name of the function to execute :type function_name: <xref:semantic_kernel.str | None> :param plugin_name: The name of the plugin to execute :type plugin_name: <xref:semantic_kernel.str | None> :param metadata: The metadata to pass to the function(s) :type metadata: <xref:semantic_kernel.dict[str, Any]> :param return_function_results: If True, the function results are yielded as a list[FunctionResult] :type return_function_results: bool :param in addition to the streaming content: :param otherwise only the streaming content is yielded.: :param kwargs: arguments that can be used instead of supplying KernelArguments :type kwargs: <xref:semantic_kernel.dict[str, Any]>

Python

```
async invoke_stream(function: KernelFunction | None = None, arguments: KernelArguments | None = None, function_name: str | None = None, plugin_name: str | None = None, metadata: dict[str, Any] = {}, return_function_results: bool = False, **kwargs: Any) -> AsyncGenerator[list['StreamingContentMixin'] | FunctionResult | list[semantic_kernel.functions.function_result.FunctionResult], Any]
```

Parameters

[Expand table](#)

Name	Description
<code>function</code>	default value: None

Name	Description
<code>arguments</code>	default value: None
<code>function_name</code>	default value: None
<code>plugin_name</code>	default value: None
<code>metadata</code>	default value: {}
<code>return_function_results</code>	default value: False

Attributes

`model_computed_fields`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

Python

```
model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}
```

`model_config`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*] [`pydantic.config.ConfigDict`].

Python

```
model_config: ClassVar[ConfigDict] = {'arbitrary_types_allowed': True,
'populate_by_name': True, 'validate_assignment': True}
```

`model_fields`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][`pydantic.fields.FieldInfo`].

This replaces *Model.fields* from Pydantic V1.

Python

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'ai_service_selector':  
    FieldInfo(annotation=AIServiceSelector, required=False,  
    default_factory=AIServiceSelector), 'auto_function_invocation_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'function_invocation_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'plugins':  
    FieldInfo(annotation=dict[str, KernelPlugin], required=False,  
    default_factory=dict), 'prompt_rendering_filters':  
    FieldInfo(annotation=list[tuple[int, Callable[list, NoneType]]],  
    required=False, default_factory=list), 'retry_mechanism':  
    FieldInfo(annotation=RetryMechanismBase, required=False,  
    default_factory=PassThroughWithoutRetry), 'services':  
    FieldInfo(annotation=dict[str, AIServiceClientBase], required=False,  
    default_factory=dict)}
```

plugins

The plugins to be used by the kernel

services

The services to be used by the kernel

ai_service_selector

The AI service selector to be used by the kernel

retry_mechanism

The retry mechanism to be used by the kernel