Aali Flowkit Python

ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Jul 18, 2025

CONTENTS

The Aali Flowkit Python is a Python service that exposes features from Aali Flowkit to Python users. This documentation provides information on how to install and use the Aali Flowkit Python.

The Aali Flowkit Python offers these main features:

Getting started Learn how to install the Aali Flowkit Python in user mode and quickly begin using it.

Getting started User guide Understand key concepts for implementing the Aali Flowkit Python in your workflow.

User guide API reference Understand how to use Python to interact programmatically with the Aali Flowkit Python.

The src.aali.flowkit library Examples Explore examples that show how to use the Aali Flowkit Python to perform many different types of operations.

Examples Contribute Learn how to contribute to the Aali Flowkit Python codebase or documentation.

Contribute

CHAPTER

ONE

GETTING STARTED

This section describes how to install the Aali Flowkit Python in user mode and quickly begin using it. If you are interested in contributing to the Aali Flowkit Python, see *Contribute* for information on installing in developer mode.

1.1 Installation

To use pip to install the Aali Flowkit Python, run this command:

```
pip install aali-flowkit-python
```

Alternatively, to install the latest version from this library's GitHub repository, run these commands:

```
git clone https://github.com/ansys/aali-flowkit-python
cd aali-flowkit-python
pip install .
```

1.1.1 Quick start

The following examples show how to use the Aali Flowkit Python.

```
aali-flowkit-python --host 0.0.0.0 --port 50052 --workers 1
```

CHAPTER

TWO

USER GUIDE

This section explains key concepts for implementing the Aali Flowkit Python in your workflow. You can use the Aali Flowkit Python in your examples as well as integrate this library into your own code.

CHAPTER

THREE

API REFERENCE

This section describes Aali Flowkit Python endpoints, their capabilities, and how to interact with them programmatically.

3.1 The `src.aali.flowkit` library

3.1.1 Summary

Subpackages

config	Configuration package for the application.
endpoints	Endpoints package responsible for defining the endpoints.
models	Models package used to define the data models.
utils	Utils module.

Submodules

__main__	Main module for the FlowKit service.
fastapi_utils	Utils module for FastAPI related operations.
flowkit_service	Module for the Aali Flowkit service.

Attributes

__version__

The `config` package

Summary

Description

Configuration package for the application.

The `endpoints` package

Summary

Submodules

mechscriptbot	Module for triggering the MechanicalScriptingBot application.
splitter	Module for splitting text into chunks.

The `mechscriptbot.py` module

Summary

Functions

triggermechscriptbot	Endpoint for triggering the MechanicalScriptingBot application.

Attributes

router

Description

Module for triggering the MechanicalScriptingBot application.

Module detail

async mechscriptbot.**triggermechscriptbot**(*request:* aali.flowkit.models.mechscriptbot.MechScriptBotRequest, *api_key: str = Header(...)*) → *aali.flowkit.models.mechscriptbot.MechScriptBotResponse*

Endpoint for triggering the MechanicalScriptingBot application.

Parameters

request
[MechScriptBotRequest] An object containing the input query and other relevant parameters for the MechanicalScriptingBot application.

api_key
[str] The API key for authentication.

Returns

MechScriptBotResponse
An object containing the output and other relevant metadata for the MechanicalScriptingBot application.

mechscriptbot.**router**

The splitter.py module

Summary

Functions

split_ppt	Endpoint for splitting text in a PowerPoint document into chunks.
split_py	Endpoint for splitting Python code into chunks.
split_pdf	Endpoint for splitting text in a PDF document into chunks.
process_ppt	Process a PowerPoint document to split text into chunks.
process_python_code	Process Python code to split text into chunks.
process_pdf	Process a PDF document to split text into chunks.
validate_request	Validate the splitter request and API key.

Attributes

router

Constants

TOKEN_TO_CHARACTER_MULTIPLIER

Description

Module for splitting text into chunks.

Module detail

async splitter.**split_ppt**(*request:* aali.flowkit.models.splitter.SplitterRequest, *api_key: str = Header(...)*) → *aali.flowkit.models.splitter.SplitterResponse*

Endpoint for splitting text in a PowerPoint document into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

api_key
[str] The API key for authentication.

async splitter.**split_py**(*request:* aali.flowkit.models.splitter.SplitterRequest, *api_key: str = Header(...)*) → *aali.flowkit.models.splitter.SplitterResponse*

Endpoint for splitting Python code into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

api_key
[str] The API key for authentication.

Returns

SplitterResponse
An object containing a list of text chunks.

async splitter.**split_pdf**(*request:* aali.flowkit.models.splitter.SplitterRequest, *api_key: str = Header(...)*) → *aali.flowkit.models.splitter.SplitterResponse*

Endpoint for splitting text in a PDF document into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'.

api_key
[str] The API key for authentication.

Returns

SplitterResponse
An object containing a list of text chunks.

splitter.**process_ppt**(*request:* aali.flowkit.models.splitter.SplitterRequest) → *aali.flowkit.models.splitter.SplitterResponse*

Process a PowerPoint document to split text into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

Returns

SplitterResponse
An object containing a list of text chunks.

splitter.**process_python_code**(*request:* aali.flowkit.models.splitter.SplitterRequest) → *aali.flowkit.models.splitter.SplitterResponse*

Process Python code to split text into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

Returns

SplitterResponse
An object containing a list of text chunks.

splitter.**process_pdf**(*request:* aali.flowkit.models.splitter.SplitterRequest) → *aali.flowkit.models.splitter.SplitterResponse*

Process a PDF document to split text into chunks.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

Returns

SplitterResponse
An object containing a list of text chunks.

splitter.**validate_request**(*request:* aali.flowkit.models.splitter.SplitterRequest, *api_key: str*)

Validate the splitter request and API key.

Parameters

request
[SplitterRequest] An object containing 'document_content' in Base64, 'chunk_size', and 'chunk_overlap'

api_key
[str] The API key for authentication.

Raises

HTTPException
If the API key is invalid or if any of the request parameters are invalid.

splitter.**TOKEN_TO_CHARACTER_MULTIPLIER = 4**

splitter.**router**

Description

Endpoints package responsible for defining the endpoints.

The `models` package

Summary

Submodules

functions	Module for defining the models used in the endpoints.
mechscriptbot	Model for the MechanicalScriptingBot endpoint.
splitter	Model for the splitter endpoint.

The `functions.py` module

Summary

Classes

ParameterInfo	Parameter information model.
EndpointInfo	Endpoint information model.

Enums

FunctionCategory	Enum for function categories.

`ParameterInfo`

class src.aali.flowkit.models.functions.**ParameterInfo**(*/*, ***data: Any*)

Bases: `pydantic.BaseModel`

Parameter information model.

Parameters

BaseModel
[`pydantic.BaseModel`] The base model for the parameter information

Overview

Attributes

name
type

Import detail

```
from src.aali.flowkit.models.functions import ParameterInfo
```

Attribute detail

ParameterInfo.**name:** str

ParameterInfo.**type:** str

EndpointInfo

class src.aali.flowkit.models.functions.**EndpointInfo**(*/*, ***data: Any*)

Bases: pydantic.BaseModel

Endpoint information model.

Parameters

BaseModel
[pydantic.BaseModel] The base model for the endpoint information

Overview

Attributes

name
path
category
display_name
description
inputs
outputs
definitions

Import detail

```
from src.aali.flowkit.models.functions import EndpointInfo
```

Attribute detail

EndpointInfo.**name**: str

EndpointInfo.**path**: str

EndpointInfo.**category**: str

EndpointInfo.**display_name**: str

EndpointInfo.**description**: str

EndpointInfo.**inputs**: list[*ParameterInfo*]

EndpointInfo.**outputs**: list[*ParameterInfo*]

EndpointInfo.**definitions**: dict[str, Any]

FunctionCategory

class src.aali.flowkit.models.functions.**FunctionCategory**(**args*, ***kwds*)

Bases: enum.Enum

Enum for function categories.

Overview

Attributes

DATA_EXTRACTION
GENERIC
KNOWLEDGE_DB
LLM_HANDLER
ANSYS_GPT

Import detail

```
from src.aali.flowkit.models.functions import FunctionCategory
```

Attribute detail

FunctionCategory.**DATA_EXTRACTION = 'data_extraction'**

FunctionCategory.**GENERIC = 'generic'**

FunctionCategory.**KNOWLEDGE_DB = 'knowledge_db'**

FunctionCategory.**LLM_HANDLER = 'llm_handler'**

FunctionCategory.**ANSYS_GPT = 'ansys_gpt'**

Description

Module for defining the models used in the endpoints.

The `mechscriptbot.py` module

Summary

Classes

MechScriptBotRequest	Request model for the MechanicalScriptingBot endpoint.
MechScriptBotResponse	Response model for the MechanicalScriptingBot endpoint.

MechScriptBotRequest

class src.aali.flowkit.models.mechscriptbot.**MechScriptBotRequest**(*/*, ***data: Any*)

Bases: pydantic.BaseModel

Request model for the MechanicalScriptingBot endpoint.

Parameters

BaseModel
[pydantic.BaseModel] The base model for the request.

Overview

Attributes

question
mech_script_bot_url
full_human_memory
full_ai_memory
full_variables
full_mechanical_objects

Import detail

```
from src.aali.flowkit.models.mechscriptbot import MechScriptBotRequest
```

Attribute detail

MechScriptBotRequest.**question**: str

MechScriptBotRequest.**mech_script_bot_url**: str

MechScriptBotRequest.**full_human_memory**: list[str]

MechScriptBotRequest.**full_ai_memory**: list[str]

MechScriptBotRequest.**full_variables**: list[str]

MechScriptBotRequest.**full_mechanical_objects**: list[str]

MechScriptBotResponse

class src.aali.flowkit.models.mechscriptbot.**MechScriptBotResponse**(*/*, ***data: Any*)

Bases: pydantic.BaseModel

Response model for the MechanicalScriptingBot endpoint.

Parameters

BaseModel
[pydantic.BaseModel] The base model for the response.

Overview

Attributes

output
updated_human_memory
updated_ai_memory
updated_variables
updated_mechanical_objects

Import detail

```
from src.aali.flowkit.models.mechscriptbot import MechScriptBotResponse
```

Attribute detail

MechScriptBotResponse.**output:** str

MechScriptBotResponse.**updated_human_memory:** list[str]

MechScriptBotResponse.**updated_ai_memory:** list[str]

MechScriptBotResponse.**updated_variables:** list[str]

MechScriptBotResponse.**updated_mechanical_objects:** list[str]

Description

Model for the MechanicalScriptingBot endpoint.

The splitter.py module

Summary

Classes

SplitterRequest	Request model for the splitter endpoint.
SplitterResponse	Response model for the splitter endpoint.

SplitterRequest

class src.aali.flowkit.models.splitter.**SplitterRequest**(*/*, ***data: Any*)

Bases: pydantic.BaseModel

Request model for the splitter endpoint.

Parameters

BaseModel
[pydantic.BaseModel] The base model for the request.

Overview

Attributes

document_content
chunk_size
chunk_overlap

Import detail

```
from src.aali.flowkit.models.splitter import SplitterRequest
```

Attribute detail

SplitterRequest.**document_content**: bytes

SplitterRequest.**chunk_size**: int

SplitterRequest.**chunk_overlap**: int

SplitterResponse

class src.aali.flowkit.models.splitter.**SplitterResponse**(*/*, ***data: Any*)

Bases: pydantic.BaseModel

Response model for the splitter endpoint.

Parameters

BaseModel
[pydantic.BaseModel] The base model for the response.

Overview

Attributes

chunks

Import detail

```
from src.aali.flowkit.models.splitter import SplitterResponse
```

Attribute detail

SplitterResponse.**chunks**: list[str]

Description

Model for the splitter endpoint.

Description

Models package used to define the data models.

The `utils` package

Summary

Submodules

decorators	Decorators module for function definitions.

The `decorators.py` module

Summary

Functions

category	Decorator to add a category to the function.
display_name	Decorator to add a display name to the function.

Description

Decorators module for function definitions.

Module detail

`decorators.`**`category`**(*value: str*)

Decorator to add a category to the function.

`decorators.`**`display_name`**(*value: str*)

Decorator to add a display name to the function.

Description

Utils module.

The `__main__.py` module

Summary

Functions

parse_cli_args	Parse the command line arguments.
handle_legacy_port_config	Handle legacy port configuration.
substitute_empty_values	Substitute the empty values with configuration values.
main	Run entrypoint for the FlowKit service.

Description

Main module for the FlowKit service.

Module detail

__main__.**parse_cli_args**()

Parse the command line arguments.

__main__.**handle_legacy_port_config**()

Handle legacy port configuration.

__main__.**substitute_empty_values**(*args*)

Substitute the empty values with configuration values.

__main__.**main**()

Run entrypoint for the FlowKit service.

The `fastapi_utils.py` module

Summary

Functions

extract_field_type	Extract the field type from a given schema field information.
extract_fields_from_schema	Extract fields and their types from a schema.
get_parameters_info	Get parameter information from function parameters.
get_return_type_info	Get return type information from the function's return type.
extract_definitions_from_schema	Extract definitions from a schema.
get_definitions_from_params	Get definitions from function parameters.
get_definitions_from_return_type	Get definitions from the function's return type.
extract_endpoint_info	Extract endpoint information from the given routes.

Description

Utils module for FastAPI related operations.

Module detail

fastapi_utils.**extract_field_type**(*field_info: dict*)

Extract the field type from a given schema field information.

Parameters

field_info
[dict] The field information from the schema.

Returns

str
The extracted field type.

fastapi_utils.**extract_fields_from_schema**(*schema: dict*)

Extract fields and their types from a schema.

Parameters

schema
[dict] The schema dictionary.

Returns

list
A list of ParameterInfo objects representing the fields.

fastapi_utils.**get_parameters_info**(*params: dict*)

Get parameter information from function parameters.

Parameters

params
[dict] A dictionary of function parameters.

Returns

list
A list of ParameterInfo objects representing the parameters.

fastapi_utils.**get_return_type_info**(*return_type: type[pydantic.BaseModel]*)

Get return type information from the function's return type.

Parameters

return_type
[type[BaseModel]] The return type of the function.

Returns

list
A list of ParameterInfo objects representing the return type fields.

fastapi_utils.**extract_definitions_from_schema**(*schema: dict*) → dict[str, Any]

Extract definitions from a schema.

Parameters

schema
[dict] The schema dictionary.

Returns

dict
A dictionary of definitions.

fastapi_utils.**get_definitions_from_params**(*params: dict*) → dict[str, Any]

Get definitions from function parameters.

Parameters

params
[dict] A dictionary of function parameters.

Returns

dict
A dictionary of definitions extracted from the parameters.

fastapi_utils.**get_definitions_from_return_type**(*return_type: type[pydantic.BaseModel]*) → dict[str, Any]

Get definitions from the function's return type.

Parameters

return_type
[type[BaseModel]] The return type of the function.

Returns

dict
A dictionary of definitions extracted from the return type.

fastapi_utils.**extract_endpoint_info**(*function_map: dict[str, Any]*, *routes: list[fastapi.routing.APIRoute]*) → list[*aali.flowkit.models.functions.EndpointInfo*]

Extract endpoint information from the given routes.

Parameters

function_map
[dict[str, Any]] A dictionary mapping function names to their implementations.

routes
[list[APIRoute]] A list of APIRoute objects representing the API routes.

Returns

list
A list of EndpointInfo objects representing the endpoints.

The `flowkit_service.py` module

Summary

Functions

list_functions	List all available functions and their endpoints.

Attributes

flowkit_service
function_map

Description

Module for the Aali Flowkit service.

Module detail

async flowkit_service.**list_functions**(*api_key: str = Header(...)*) → list[*aali.flowkit.models.functions.EndpointInfo*]

List all available functions and their endpoints.

Parameters

api_key
[str] The API key for authentication.

Returns

list[EndpointInfo]
A list of EndpointInfo objects representing the endpoints.

flowkit_service.**flowkit_service**

flowkit_service.**function_map**

3.1.2 Description

App package responsible for creating the FastAPI app.

3.1.3 Module detail

flowkit.__version__

CHAPTER

FOUR

EXAMPLES

This section show how to use the Aali Flowkit Python to perform many different types of operations.

CHAPTER

FIVE

CONTRIBUTE

Overall guidance on contributing to a PyAnsys library appears in the Contributing topic in the *PyAnsys developer's guide*. Ensure that you are thoroughly familiar with this guide before attempting to contribute to the Aali Flowkit Python.

The following contribution information is specific to the Aali Flowkit Python.

5.1 Clone the repository

To clone and install the latest *Aali Flowkit Python* release in development mode, run these commands:

```
git clone https://github.com/ansys/aali-flowkit-python/
cd aali-flowkit-python
python -m pip install --upgrade pip
pip install -e .
```

5.2 Adhere to code style

Aali Flowkit Python follows the PEP8 standard as outlined in PEP 8 in the PyAnsys Developer's Guide and implements style checking using pre-commit.

To ensure your code meets minimum code styling standards, run these commands:

```
pip install pre-commit
pre-commit run --all-files
```

You can also install this as a pre-commit hook by running this command:

```
pre-commit install
```

5.3 Run the tests

Prior to running the tests, you must run this command to install the test dependencies:

```
pip install -e .[tests]
```

To run the tests, navigate to the root directory of the repository and run this command:

```
pytest
```

5.4 Build the documentation

Prior to building the documentation, you must run this command to install the documentation dependencies:

```
pip install -e .[doc]
```

To build the documentation, run the following commands:

```
cd doc

# On linux
make html

# On windows
./make.bat html
```

The documentation is built in the *docs/_build/html* directory.

PYTHON MODULE INDEX

S

INDEX

Symbols

A

C

D

E

F

G

H

I

K

L

M