

© 2025 ANSYS, Inc. or affiliated companies Unauthorized use, distribution, or duplication prohibited.

# **PyAnsys Geometry**



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

Feb 28, 2025

ANSYS, Inc. and ANSYS Europe, Ltd. are UL registered ISO 9001:2015 companies.

# **CONTENTS**



PyAnsys Geometry is a Python client library for the Ansys Geometry service. You are looking at the documentation for version 0.9.1.

Getting started Learn how to run the Windows Docker container, install the PyAnsys Geometry image, and launch and connect to the Geometry service.

Getting started User guide Understand key concepts and approaches for primitives, sketches, and model designs.

*User guide* API reference Understand PyAnsys Geometry API endpoints, their capabilities, and how to interact with them programmatically.

API reference Examples Explore examples that show how to use PyAnsys Geometry to perform many different types of operations.

Examples Contribute Learn how to contribute to the PyAnsys Geometry codebase or documentation.

*Contribute* Assets Download different assets related to PyAnsys Geometry, such as documentation, package wheelhouse, and related files.

Assets

PyAnsys Geometry is a Python client library for the Ansys Geometry service.

CONTENTS 1

2 CONTENTS

### **CHAPTER**

# ONE

## **INSTALLATION**

You can use pip to install PyAnsys Geometry.

pip install ansys-geometry-core

### **CHAPTER**

### **TWO**

### **AVAILABLE MODES**

This client library works with a Geometry service backend. There are several ways of running this backend, although the preferred and high-performance mode is using Docker containers. Select the option that suits your needs best.

Docker containers Launch the Geometry service as a Docker container and connect to it from PyAnsys Geometry.

Docker containers Local service Launch the Geometry service locally on your machine and connect to it from PyAnsys Geometry.

*Launch a local session* Remote service Launch the Geometry service on a remote machine and connect to it using PIM (Product Instance Manager).

Launch a remote session Connect to an existing Service Connect to an existing Geometry service locally or remotely.

Use an existing session

## **THREE**

## **COMPATIBILITY WITH ANSYS RELEASES**

PyAnsys Geometry continues to evolve as the Ansys products move forward. For more information, see *Ansys product version compatibility*.

_		_			_
~			РΊ	гг	R
	н	Δ	$\mathbf{r}$	-	ĸ

# **FOUR**

## **DEVELOPMENT INSTALLATION**

In case you want to support the development of PyAnsys Geometry, install the repository in development mode. For more information, see *Install package in development mode*.

**CHAPTER** 

**FIVE** 

### FREQUENTLY ASKED QUESTIONS

Any questions? Refer to Q&A before submitting an issue.

### 5.1 Docker containers

### 5.1.1 What is Docker?

Docker is an open platform for developing, shipping, and running apps in a containerized way.

Containers are standard units of software that package the code and all its dependencies so that the app runs quickly and reliably from one computing environment to another.

Ensure that the machine where the Geometry service is to run has Docker installed. Otherwise, see Install Docker Engine in the Docker documentation.

### 5.1.2 Select your Docker container

Currently, the Geometry service backend is mainly delivered as a **Windows** Docker container. However, these containers require a Windows machine to run them.

Select the kind of Docker container you want to build:

Windows Docker container Build a Windows Docker container for the Geometry service and use it from PyAnsys Geometry. Explore the full potential of the Geometry service.

Windows Docker container

Go to Getting started

#### **Windows Docker container**

#### Contents

- Windows Docker container
  - Docker for Windows containers
  - Build or install the Geometry service image
    - \* GitHub Container Registry
    - \* Build the Geometry service Windows container
      - · Prerequisites
      - · Build from available Ansys installation

- · Build the Docker image from available binaries
- Launch the Geometry service
  - \* Environment variables
  - \* Geometry service launcher
- Connect to the Geometry service

#### **Docker for Windows containers**

To run the Windows Docker container for the Geometry service, ensure that you follow these steps when installing Docker:

- 1. Install Docker Desktop.
- 2. When prompted for **Use WSL2 instead of Hyper-V** (**recommended**), **clear** this checkbox. Hyper-V must be enabled to run Windows Docker containers.
- 3. Once the installation finishes, restart your machine and start Docker Desktop.
- 4. On the Windows taskbar, go to the **Show hidden icons** section, right-click in the Docker Desktop app, and select **Switch to Windows containers**.

Now that your Docker engine supports running Windows Docker containers, you can build or install the PyAnsys Geometry image.

### **Build or install the Geometry service image**

There are two options for installing the PyAnsys Geometry image:

- Download it from the *GitHub Container Registry*.
- Build the Geometry service Windows container.

### **GitHub Container Registry**



This option is only available for users with write access to the repository or who are members of the Ansys organization.

Once Docker is installed on your machine, follow these steps to download the Windows Docker container for the Geometry service and install this image.

- 1. Using your GitHub credentials, download the Docker image from the PyAnsys Geometry repository on GitHub.
- 2. Use a GitHub personal access token with permission for reading packages to authorize Docker to access this repository. For more information, see Managing your personal access tokens in the GitHub documentation.
- 3. Save the token to a file with this command:

### 

4. Authorize Docker to access the repository and run the commands for your OS. To see these commands, click the tab for your OS.

### **Powershell**

```
$env:GH_USERNAME=<my-github-username>
cat GH_TOKEN.txt | docker login ghcr.io -u $env:GH_USERNAME --password-stdin
```

### **Windows CMD**

```
SET GH_USERNAME=<my-github-username>
type GH_TOKEN.txt | docker login ghcr.io -u %GH_USERNAME% --password-stdin
```

5. Pull the Geometry service locally using Docker with a command like this:

```
docker pull ghcr.io/ansys/geometry:windows-latest
```

### **Build the Geometry service Windows container**

The Geometry service Docker containers can be easily built by following these steps.

Inside the repository's docker folder, there are two Dockerfile files:

- linux/Dockerfile: Builds the Linux-based Docker image.
- windows/Dockerfile: Builds the Windows-based Docker image.

Depending on the characteristics of the Docker engine installed on your machine, either one or the other has to be built.

This guide focuses on building the windows/Dockerfile image.

There are two build modes:

- Build from available Ansys installation: This mode builds the Docker image using the Ansys installation available in the machine where the Docker image is being built.
- Build from available binaries: This mode builds the Docker image using the binaries available in the ansys/pyansys-geometry-binaries repository. If you do not have access to this repository, you can only use the first mode.

### **Prerequisites**

Ensure that Docker is installed in your machine. If you do not have Docker available, see Docker for Windows
containers.

### **Build from available Ansys installation**

To build your own image based on your own Ansys installation, follow these instructions:

- Download the Python Docker build script.
- Execute the script with the following command (no specific location needed):

```
python build_docker_windows.py
```

Check that the image has been created successfully. You should see output similar to this:

5.1. Docker containers 13

(continued from previous page)

### **Build the Docker image from available binaries**

Prior to building your image, follow these steps:

- Download the latest Windows Dockerfile.
- Download the latest release artifacts for the Windows Docker container (ZIP file) for your version.



Only users with access to https://github.com/ansys/pyansys-geometry-binaries can download these binaries.

• Move this ZIP file to the location of the Windows Dockerfile previously downloaded.

To build your image, follow these instructions:

- 1. Navigate to the folder where the ZIP file and Dockerfile are located.
- 2. Run this Docker command:

```
docker build -t ghcr.io/ansys/geometry:windows-latest -f windows/Dockerfile .
```

3. Check that the image has been created successfully. You should see output similar to this:

### Launch the Geometry service

There are methods for launching the Geometry service:

- You can use the PyAnsys Geometry launcher.
- You can manually launch the Geometry service.

### **Environment variables**

The Geometry service requires this mandatory environment variable for its use:

• LICENSE\_SERVER: License server (IP address or DNS) that the Geometry service is to connect to. For example, 127.0.0.1.

You can also specify other optional environment variables:

• ENABLE\_TRACE: Whether to set up the trace level for debugging purposes. The default is 0, in which case the trace level is not set up. Options are 1 and 0.

• LOG\_LEVEL: Sets the Geometry service logging level. The default is 2, in which case the logging level is INFO.

Here are some terms to keep in mind:

- host: Machine that hosts the Geometry service. It is typically on localhost, but if you are deploying the service on a remote machine, you must pass in this host machine's IP address when connecting. By default, PyAnsys Geometry assumes it is on localhost.
- **port**: Port that exposes the Geometry service on the host machine. Its value is assumed to be 50051, but users can deploy the service on preferred ports.

Prior to using the PyAnsys Geometry launcher to launch the Geometry service, you must define general environment variables required for your OS. You do not need to define these environment variables prior to manually launching the Geometry service.

### **Using PyAnsys Geometry launcher**

Define the following general environment variables prior to using the PyAnsys Geometry launcher. Click the tab for your OS to see the appropriate commands.

### Linux/Mac

```
export ANSRV_GEO_LICENSE_SERVER=127.0.0.1
export ANSRV_GEO_ENABLE_TRACE=0
export ANSRV_GEO_LOG_LEVEL=2
export ANSRV_GEO_HOST=127.0.0.1
export ANSRV_GEO_PORT=50051
```

### **Powershell**

```
$env:ANSRV_GEO_LICENSE_SERVER="127.0.0.1"
$env:ANSRV_GEO_ENABLE_TRACE=0
$env:ANSRV_GEO_LOG_LEVEL=2
$env:ANSRV_GEO_HOST="127.0.0.1"
$env:ANSRV_GEO_PORT=50051
```

#### Windows CMD

```
SET ANSRV_GEO_LICENSE_SERVER=127.0.0.1
SET ANSRV_GEO_ENABLE_TRACE=0
SET ANSRV_GEO_LOG_LEVEL=2
SET ANSRV_GEO_HOST=127.0.0.1
SET ANSRV_GEO_PORT=50051
```

### Warning

When running a Windows Docker container, certain high-value ports might be restricted from its use. This means that the port exposed by the container has to be set to lower values. You should change the value of ANSRV\_GEO\_PORT to use a port such as 700, instead of 50051.

5.1. Docker containers 15

### **Manual launch**

You do not need to define general environment variables prior to manually launching the Geometry service. They are directly passed to the Docker container itself.

### **Geometry service launcher**

As mentioned earlier, you can launch the Geometry service locally in two different ways. To see the commands for each method, click the following tabs.

### **Using PyAnsys Geometry launcher**

This method directly launches the Geometry service and provides a Modeler object.

```
from ansys.geometry.core.connection import launch_modeler
modeler = launch_modeler()
```

The launch\_modeler() method launches the Geometry service under the default conditions. For more configurability, use the launch\_docker\_modeler() method.

#### **Manual launch**

This method requires that you manually launch the Geometry service. Remember to pass in the different environment variables that are needed. Afterwards, see the next section to understand how to connect to this service instance from PyAnsys Geometry.

#### Linux/Mac

```
docker run \
    --name ans_geo \
    -e LICENSE_SERVER=<LICENSE_SERVER> \
    -p 50051:50051 \
    ghcr.io/ansys/geometry:<TAG>
```

### **Powershell**

```
docker run `
    --name ans_geo `
    -e LICENSE_SERVER=<LICENSE_SERVER> `
    -p 50051:50051 `
    ghcr.io/ansys/geometry:<TAG>
```

#### Windows CMD

```
docker run ^
    --name ans_geo ^
    -e LICENSE_SERVER=<LICENSE_SERVER> ^
    -p 50051:50051 ^
    ghcr.io/ansys/geometry:<TAG>
```

### 🛕 Warning

When running a Windows Docker container, certain high-value ports might be restricted from its use. This means that the port exposed by the container has to be set to lower values. You should change the value of -p 50051: 50051 to use a port such as -p 700:50051.

### **Connect to the Geometry service**

After the Geometry service is launched, connect to it with these commands:

```
from ansys.geometry.core import Modeler
modeler = Modeler()
```

By default, the Modeler instance connects to 127.0.0.1 ("localhost") on port 50051. You can change this by modifying the host and port parameters of the Modeler object, but note that you must also modify your docker run command by changing the <HOST-PORT>-50051 argument.

The following tabs show the commands that set the environment variables and Modeler function.

### Warning

When running a Windows Docker container, certain high-value ports might be restricted from its use. This means that the port exposed by the container has to be set to lower values. You should change the value of ANSRV\_GEO\_PORT to use a port such as 700, instead of 50051.

### **Environment variables**

### Linux/Mac

```
export ANSRV_GEO_HOST=127.0.0.1
export ANSRV_GEO_PORT=50051
```

#### **Powershell**

```
$env:ANSRV_GEO_HOST="127.0.0.1"
$env:ANSRV_GEO_PORT=50051
```

#### Windows CMD

```
SET ANSRV_GEO_HOST=127.0.0.1
SET ANSRV GEO PORT=50051
```

### **Modeler function**

```
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler(host="127.0.0.1", port=50051)
```

Go to Docker containers

Go to Getting started

5.1. Docker containers

### 5.2 Launch a local session

If Ansys 2024 R1 or later and PyAnsys Geometry are installed, you can create a local backend session using Discovery, SpaceClaim, or the Geometry service. Once the backend is running, PyAnsys Geometry can manage the connection.

To launch and establish a connection to the service, open Python and use the following commands for either Discovery, SpaceClaim, or the Geometry service.

### **Discovery**

```
from ansys.geometry.core import launch_modeler_with_discovery

modeler = launch_modeler_with_discovery()
```

### **SpaceClaim**

```
from ansys.geometry.core import launch_modeler_with_spaceclaim

modeler = launch_modeler_with_spaceclaim()
```

### **Geometry service**

```
from ansys.geometry.core import launch_modeler_with_geometry_service
modeler = launch_modeler_with_geometry_service()
```

When launching via Geometry Service, if you have a custom local install, you can define the path of this install in the ANSYS\_GEOMETRY\_SERVICE\_ROOT environment variable. In that case, the launcher uses this location by default.

#### **Powershell**

```
$env:ANSYS_GEOMETRY_SERVICE_ROOT="C:\Program Files\ANSYS Inc\v252\GeometryService"
# or
$env:ANSYS_GEOMETRY_SERVICE_ROOT="C:\myCustomPath\CoreGeometryService"
```

### **Windows CMD**

```
SET ANSYS_GEOMETRY_SERVICE_ROOT="C:\Program Files\ANSYS Inc\v252\GeometryService"
# or
SET ANSYS_GEOMETRY_SERVICE_ROOT="C:\myCustomPath\CoreGeometryService"
```

### Linux

```
export ANSYS_GEOMETRY_SERVICE_ROOT=/my_path/to/core_geometry_service
```

For more information on the arguments accepted by the launcher methods, see their API documentation:

- launch\_modeler\_with\_discovery
- launch\_modeler\_with\_spaceclaim
- launch\_modeler\_with\_geometry\_service



Because this is the first release of the Geometry service, you cannot yet define a product version or API version.

Go to Getting started

### 5.3 Launch a remote session

If a remote server is running Ansys 2024 R1 or later and is also running PIM (Product Instance Manager), you can use PIM to start a Discovery or SpaceClaim session that PyAnsys Geometry can connect to.



### Warning

This option is only available for Ansys employees.

Only Ansys employees with credentials to the Artifact Repository Browser can download ZIP files for PIM.

### 5.3.1 Set up the client machine

1. To establish a connection to the existing session from your client machine, open Python and run these commands:

```
from ansys.discovery.core import launch_modeler_with_pimlight_and_discovery
disco = launch_modeler_with_pimlight_and_discovery("241")
```

The preceding commands launch a Discovery (version 24.1) session with the API server. You receive a model object back from Discovery that you then use as a PyAnsys Geometry client.

2. Start SpaceClaim or the Geometry service remotely using commands like these:

```
from ansys.discovery.core import launch_modeler_with_pimlight_and_spaceclaim
sc = launch_modeler_with_pimlight_and_spaceclaim("version")
from ansys.discovery.core import launch_modeler_with_pimlight_and_geometry_service
geo = launch_modeler_with_pimlight_and_geometry_service("version")
```



### 1 Note

Performing all these operations remotely eliminates the need to worry about the starting endpoint or managing the session.

### 5.3.2 End the session

To end the session, run the corresponding command:

```
disco.close()
sc.close()
geo.close()
```

Go to Getting started

### 5.4 Use an existing session

If a session of Discovery, SpaceClaim, or the Geometry service is already running, PyAnsys Geometry can be used to connect to it.



### 🛕 Warning

Running a SpaceClaim or Discovery normal session does not suffice to be able to use it with PyAnsys Geometry. Both products need the ApiServer extension to be running. In this case, to ease the process, you should launch the products directly from the PyAnsys Geometry library as shown in Launch a local session.

### 5.4.1 Establish the connection

From Python, establish a connection to the existing client session by creating a Modeler object:

```
from ansys.geometry.core import Modeler
modeler = Modeler(host="localhost", port=50051)
```

If no error messages are received, your connection is established successfully. Note that your local port number might differ from the one shown in the preceding code.

### 5.4.2 Verify the connection

If you want to verify that the connection is successful, request the status of the client connection inside your Modeler object:

```
>>> modeler.client
Ansys Geometry Modeler Client (...)
Target:
           localhost:50051
Connection: Healthy
```

Go to Getting started

### 5.5 Ansys version compatibility

The following table summarizes the compatibility matrix between the PyAnsys Geometry service and the Ansys product versions.

PyAnsys Geometry versions	Ansys Product versions	Geometry Service (dockerized)	Geometry Service (standalone)	Dis- cov- ery	Space- Claim
0.2.X	23R2				
0.3.X	23R2 (partially)				
0.4.X	24R1 onward				
0.5.X	24R1 onward				

### Ţip

Forth- and back-compatibility mechanism

Starting on version 0.5.X and onward, PyAnsys Geometry has implemented a forth- and back-compatibility mechanism to ensure that the Python library can be used with different versions of the Ansys products.

Methods are now decorated with the <code>@min\_backend\_version</code> decorator to indicate the compatibility with the Ansys product versions. If an unsupported method is called, a <code>GeometryRuntimeError</code> is raised when attempting to use the method. Users are informed of the minimum Ansys product version required to use the method.

Access to the documentation for the preceding versions is found at the Versions page.

Go to Getting started

### 5.6 Install package in development mode

This topic assumes that you want to install PyAnsys Geometry in developer mode so that you can modify the source and enhance it. You can install PyAnsys Geometry from PyPI, Conda, or from the PyAnsys Geometry repository on GitHub.

#### **Contents**

- Install package in development mode
  - Package dependencies
  - PyPI
  - Conda
  - GitHub
  - Install in offline mode
  - Verify your installation

### 5.6.1 Package dependencies

PyAnsys Geometry is supported on Python version 3.10 and later. As indicated in the Moving to require Python 3 statement, previous versions of Python are no longer supported.

PyAnsys Geometry dependencies are automatically checked when packages are installed. These projects are required dependencies for PyAnsys Geometry:

- ansys-api-geometry: Used for supplying gRPC code generated from Protobuf (PROTO) files
- NumPy: Used for data array access
- · Pint: Used for measurement units
- PyVista: Used for interactive 3D plotting
- SciPy: Used for geometric transformations

### 5.6.2 PyPI

Before installing PyAnsys Geometry, to ensure that you have the latest version of pip, run this command:

```
python -m pip install -U pip
```

Then, to install PyAnsys Geometry, run this command:

```
python -m pip install ansys-geometry-core
```

### 5.6.3 Conda

You can also install PyAnsys Geometry using conda. First, ensure that you have the latest version:

```
conda update -n base -c defaults conda
```

Then, to install PyAnsys Geometry, run this command:

```
conda install -c conda-forge ansys-geometry-core
```

#### 5.6.4 GitHub

To install the latest release from the PyAnsys Geometry repository on GitHub, run these commands:

```
git clone https://github.com/ansys/pyansys-geometry
cd pyansys-geometry
pip install -e .
```

To verify your development installation, run this command:

tox

### 5.6.5 Install in offline mode

If you lack an internet connection on your installation machine (or you do not have access to the private Ansys PyPI packages repository), you should install PyAnsys Geometry by downloading the wheelhouse archive for your corresponding machine architecture from the repository's Releases page.

Each wheelhouse archive contains all the Python wheels necessary to install PyAnsys Geometry from scratch on Windows, Linux, and MacOS from Python 3.10 to 3.13. You can install this on an isolated system with a fresh Python installation or on a virtual environment.

For example, on Linux with Python 3.10, unzip the wheelhouse archive and install it with these commands:

```
unzip ansys-geometry-core-v0.9.1-all-wheelhouse-ubuntu-3.10.zip wheelhouse pip install ansys-geometry-core -f wheelhouse --no-index --upgrade --ignore-installed
```

If you are on Windows with Python 3.10, unzip the wheelhouse archive to a wheelhouse directory and then install using the same pip install command as in the preceding example.

Consider installing using a virtual environment. For more information, see Creation of virtual environments in the Python documentation.

### 5.6.6 Verify your installation

Verify the Modeler() connection with this code:

```
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> print(modeler)

Ansys Geometry Modeler (0x205c5c17d90)

Ansys Geometry Modeler Client (0x205c5c16e00)
Target: localhost:652
Connection: Healthy
```

If you see a response from the server, you can start using PyAnsys Geometry as a service. For more information on PyAnsys Geometry usage, see *User guide*.

Go to Getting started

### 5.7 Frequently asked questions

### 5.7.1 What is PyAnsys?

PyAnsys is a set of open source Python libraries that allow you to interface with Ansys Electronics Desktop (AEDT), Ansys Mechanical, Ansys Parametric Design Language (APDL), Ansys Fluent, and other Ansys products.

You can use PyAnsys libraries within a Python environment of your choice in conjunction with external Python libraries.

### 5.7.2 How is the Ansys Geometry Service installed?



 $This\ question\ is\ answered\ in\ https://github.com/ansys/pyansys-geometry/issues/1022\ and\ https://github.com/ansys/pyansys-geometry/discussions/883$ 

The Ansys Geometry service is available as a standalone service and it is installed through the Ansys unified installer or the automated installer. Both are available for download from the Ansys Customer Portal.

When using the unified or automated installer, it is necessary to pass in the -geometryservice flag to install it.

Overall, the command to install the Ansys Geometry service with the unified installer is:

```
setup.exe -silent -geometryservice
```

You can verify that the installation was successful by checking whether the product has been installed on your file directory. If you are using the default installation directory, the product is installed in the following directory:

```
C:\Program Files\ANSYS Inc\vXXX\GeometryService
```

Where vXXX is the Ansys version that you have installed.

### 5.7.3 What Ansys license is needed to run the Geometry service?

### 1 Note

This question is answered in https://github.com/ansys/pyansys-geometry/discussions/754.

The Ansys Geometry service is a headless service developed on top of the modeling libraries for Discovery and Space-Claim.

Both in its standalone and Docker versions, the Ansys Geometry service requires a Discovery Modeling license to run.

To run PyAnsys Geometry against other backends, such as Discovery or SpaceClaim, users must have an Ansys license that allows them to run these Ansys products.

The **Discovery Modeling** license is one of these licenses, but there are others, such as the Ansys Mechanical Enterprise license, that also allow users to run these Ansys products. However, the Geometry service is only compatible with the Discovery Modeling license.

### 5.7.4 How to build the Docker image for the Ansys Geometry service?



### 1 Note

This question is answered in https://github.com/ansys/pyansys-geometry/discussions/883

To build your own Docker image for the Ansys Geometry service, users should follow the instructions provided in Build from available Ansys installation. The resulting image is a Windows-based Docker image that contains the Ansys Geometry service.

Go to Getting started

**CHAPTER** 

SIX

### **USER GUIDE**

This section provides an overview of the PyAnsys Geometry library, explaining key concepts and approaches for primitives, sketches (2D basic shape elements), and model designs.

### 6.1 Primitives

The PyAnsys Geometry *math* subpackage consists of primitive representations of basic geometric objects, such as a point, vector, and matrix. To operate and manipulate physical quantities, this subpackage uses Pint, a third-party open source software that other PyAnsys libraries also use. It also uses its *shapes* subpackage to evaluate and represent geometric shapes (both curves and surfaces), such as lines, circles, cones, spheres and torus.

This table shows PyAnsys Geometry names and base values for the physical quantities:

Name	value
LENGTH_ACCURACY	1e-8
ANGLE_ACCURACY	1e-6
DEFAULT_UNITS.LENGTH	meter
DEFAULT_UNITS.ANGLE	radian

To define accuracy and measurements, you use these PyAnsys Geometry classes:

- Accuracy()
- Measurements()

### 6.1.1 Planes

The *Plane()* class provides primitive representation of a 2D plane in 3D space. It has an origin and a coordinate system. Sketched shapes are always defined relative to a plane. The default working plane is XY, which has (0,0) as its origin.

If you create a 2D object in the plane, PyAnsys Geometry converts it to the global coordinate system so that the 2D feature executes as expected:

```
from ansys.geometry.core.math import Plane, Point3D, UnitVector3D

origin = Point3D([42, 99, 13])
plane = Plane(origin, UnitVector3D([1, 0, 0]), UnitVector3D([0, 1, 0]))
```

### 6.2 Sketch

The PyAnsys Geometry *sketch* subpackage is used to build 2D basic shapes. Shapes consist of two fundamental constructs:

- Edge: A connection between two or more 2D points along a particular path. An edge represents an open shape such as an arc or line.
- Face: A set of edges that enclose a surface. A face represents a closed shape such as a circle or triangle.

To initialize a sketch, you first specify the *Plane()* class, which represents the plane in space from which other PyAnsys Geometry objects can be located.

This code shows how to initialize a sketch:

```
from ansys.geometry.core.sketch import Sketch
sketch = Sketch()
```

You then construct a sketch, which can be done using different approaches.

### 6.2.1 Functional-style API

A functional-style API is sometimes called a *fluent functional-style api* or *fluent API* in the developer community. However, to avoid confusion with the Ansys Fluent product, the PyAnsys Geometry documentation refrains from using the latter terms.

One of the key features of a functional-style API is that it keeps an active context based on the previously created edges to use as a reference starting point for additional objects.

The following code creates a sketch with its origin as a starting point. Subsequent calls create segments, which take as a starting point the last point of the previous edge.

```
from ansys.geometry.core.math import Point2D

sketch.segment_to_point(Point2D([3, 3]), "Segment2").segment_to_point(
    Point2D([3, 2]), "Segment3"
)
sketch.plot()
```

A functional-style API is also able to get a desired shape of the sketch object by taking advantage of user-defined labels:

```
sketch.get("Segment2")
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...
```

### 6.2.2 Direct API

A direct API is sometimes called an *element-based approach* in the developer community.

This code shows how you can use a direct API to create multiple elements independently and combine them all together in a single plane:

(continued from previous page)

For more information on sketch shapes, see the Sketch() subpackage.

### 6.3 Designer

The PyAnsys Geometry *designer* subpackage organizes geometry assemblies and synchronizes to a supporting Geometry service instance.

### 6.3.1 Create the model

This code create the Modeler() object which owns the whole designs tools and data.

```
from ansys.geometry.core import Modeler

# Create the modeler object itself
modeler = Modeler()
```

### 6.3.2 Define the model

The following code define the model by creating a sketch with a circle on the client. It then creates the model on the server.

```
from ansys.geometry.core.sketch import Sketch
from ansys.geometry.core.math import Point2D
from ansys.geometry.core.misc import UNITS
from pint import Quantity

# Create a sketch and draw a circle on the client
sketch = Sketch()
sketch.circle(Point2D([10, 10], UNITS.mm), Quantity(10, UNITS.mm))

# Create your design on the server
design_name = "ExtrudeProfile"
design = modeler.create_design(design_name)
```

### 6.3.3 Add materials to model

This code adds the data structure and properties for individual materials:

```
from ansys.geometry.core.materials.material import Material
from ansys.geometry.core.materials.property import (
    MaterialProperty,
    MaterialPropertyType,
)
density = Quantity(125, 1000 * UNITS.kg / (UNITS.m * UNITS.m * UNITS.m))
```

6.3. Designer 27

(continues on next page)

(continued from previous page)

```
poisson_ratio = Quantity(0.33, UNITS.dimensionless)
tensile_strength = Quantity(45)
material = Material(
    "steel",
    density,
    [MaterialProperty(MaterialPropertyType.POISSON_RATIO, "myPoisson", poisson_ratio)],
)
material.add_property(MaterialPropertyType.TENSILE_STRENGTH, "myTensile", Quantity(45))
design.add_material(material)
```

### 6.3.4 Create bodies by extruding the sketch

Extruding a sketch projects all of the specified geometries onto the body. To create a solid body, this code extrudes the sketch profile by a given distance.

```
body = design.extrude_sketch("JustACircle", sketch, Quantity(10, UNITS.mm))
```

### 6.3.5 Create bodies by extruding the face

The following code shows how you can also extrude a face profile by a given distance to create a solid body. There are no modifications against the body containing the source face.

```
longer_body = design.extrude_face(
    "LongerCircleFace", body.faces[0], Quantity(20, UNITS.mm)
)
```

You can also translate and tessellate design bodies and project curves onto them. For more information, see these classes:

- *Body()*
- Component()

### 6.3.6 Download and save design

You can save your design to disk or download the design of the active Geometry server instance. The following code shows how to download and save the design.

```
file = "path/to/download.scdocx"
design.download(file)
```

For more information, see the *Design* submodule.

### 6.4 PyAnsys Geometry overview

PyAnsys Geometry is a Python wrapper for the Ansys Geometry service. Here are some of the key features of PyAnsys Geometry:

- Ability to use the library alongside other Python libraries
- A functional-style API for a clean and easy coding experience
- · Built-in examples

### 6.5 Simple interactive example

This simple interactive example shows how to start an instance of the Geometry server and create a geometry model.

### 6.5.1 Start Geometry server instance

The <code>Modeler()</code> class within the <code>ansys-geometry-core</code> library creates an instance of the Geometry service. By default, the <code>Modeler</code> instance connects to 127.0.0.1 ("localhost") on port 50051. You can change this by modifying the host and port parameters of the <code>Modeler</code> object, but note that you must also modify your docker run command by changing the <code><HOST-PORT>:50051</code> argument.

This code starts an instance of the Geometry service:

```
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
```

### 6.5.2 Create geometry model

Once an instance has started, you can create a geometry model by initializing the *sketch* subpackage and using the *shapes* subpackage.

```
from ansys.geometry.core.math import Plane, Point3D, Point2D
from ansys.geometry.core.misc import UNITS
from ansys.geometry.core.sketch import Sketch

# Define our sketch
origin = Point3D([0, 0, 10])
plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 1, 0])

# Create the sketch
sketch = Sketch(plane)
sketch.circle(Point2D([1, 1]), 30 * UNITS.m)
sketch.plot()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...
```

**CHAPTER** 

### **SEVEN**

### **API REFERENCE**

This section describes ansys-geometry-core endpoints, their capabilities, and how to interact with them programmatically.

## 7.1 The ansys.geometry.core library

### **7.1.1 Summary**

### **Subpackages**

connection	PyAnsys Geometry connection subpackage.
designer	PyAnsys Geometry designer subpackage.
materials	PyAnsys Geometry materials subpackage.
math	PyAnsys Geometry math subpackage.
misc	Provides the PyAnsys Geometry miscellaneous subpackage.
parameters	PyAnsys Geometry parameters subpackage.
plotting	Provides the PyAnsys Geometry plotting subpackage.
shapes	Provides the PyAnsys Geometry geometry subpackage.
sketch	PyAnsys Geometry sketch subpackage.
tools	PyAnsys Geometry tools subpackage.

### **Submodules**

errors	Provides PyAnsys Geometry-specific errors.
logger	Provides a general framework for logging in PyAnsys Geometry.
modeler	Provides for interacting with the Geometry service.
typing	Provides typing of values for PyAnsys Geometry.

### **Attributes**

\_\_version\_ PyAnsys Geometry version.

### **Constants**

USE_SERVICE_COLORS	Global constant for checking whether to use service colors for plotting
DISABLE_MULTIPLE_DESIGN_CHECK	Deprecated constant. Use DISABLE_ACTIVE_DESIGN_CHECK instead.
DISABLE_ACTIVE_DESIGN_CHECK	Global constant for disabling the ensure_design_is_active check.
DOCUMENTATION_BUILD	Global flag for the documentation to use the proper PyVista Jupyter backend.
	end.

### The connection package

### **Summary**

### **Submodules**

backend	Module providing definitions for the backend types.
client	Module providing a wrapped abstraction of the gRPC stubs.
conversions	Module providing for conversions.
defaults	Module providing default connection parameters.
docker_instance	Module for connecting to a local Geometry Service Docker container.
launcher	Module for connecting to instances of the Geometry service.
<pre>product_instance</pre>	Module containing the ProductInstance class.
validate	Module to perform a connection validation check.

### The backend.py module

### **Summary**

### **Enums**

BackendType	Provides an enum holding the available backend types.
<b>ApiVersions</b>	Provides an enum for all the compatibles API versions.

### ${\bf BackendType}$

class ansys.geometry.core.connection.backend.BackendType(\*args, \*\*kwds)

Bases: enum.Enum

Provides an enum holding the available backend types.

### **Overview**

### **Attributes**

DISCOVERY
SPACECLAIM
WINDOWS\_SERVICE
LINUX\_SERVICE
CORE\_WINDOWS
CORE\_LINUX
DISCOVERY\_HEADLESS

### Static methods

*is\_core\_service* Determine whether the backend is CoreService based or not.

### Import detail

from ansys.geometry.core.connection.backend import BackendType

### Attribute detail

```
BackendType.DISCOVERY = 0
BackendType.SPACECLAIM = 1
BackendType.WINDOWS_SERVICE = 2
BackendType.LINUX_SERVICE = 3
BackendType.CORE_WINDOWS = 4
BackendType.CORE_LINUX = 5
BackendType.DISCOVERY_HEADLESS = 6
```

### **Method detail**

 $\textbf{static} \ \ \textbf{BackendType.is\_core\_service}(\textit{backend\_type:} \ \ \textbf{BackendType}) \rightarrow \textbf{bool}$ 

Determine whether the backend is CoreService based or not.

### **Parameters**

### backend\_type

[BackendType] The backend type to check whether or not it's a CoreService type.

### Returns

bool

True if the backend is CoreService based, False otherwise.

### **ApiVersions**

```
\textbf{class} \ \texttt{ansys.geometry.core.connection.backend.ApiVersions} (*\textit{args}, **\textit{kwds})
```

Bases: enum. Enum

Provides an enum for all the compatibles API versions.

#### Overview

## **Attributes**

<i>V</i> _	21	
<i>V</i> _	22	
<i>V</i> _	231	
<i>V</i> _	232	
<i>V</i> _	241	
<i>V</i> _	242	
<i>V</i> _	251	

## Static methods

parse\_input Convert an input to an ApiVersions enum.

# Import detail

from ansys.geometry.core.connection.backend import ApiVersions

## **Attribute detail**

```
ApiVersions.V_21 = 21
ApiVersions.V_22 = 22
ApiVersions.V_231 = 231
ApiVersions.V_232 = 232
ApiVersions.V_241 = 241
ApiVersions.V_242 = 242
ApiVersions.V_251 = 251
```

## Method detail

static ApiVersions.parse\_input( $version: int \mid str \mid$  ApiVersions)  $\rightarrow$  ApiVersions Convert an input to an ApiVersions enum.

# **Parameters**

## version

[int | str | ApiVersions] The version to convert to an ApiVersions enum.

# Returns

## **ApiVersions**

The version as an ApiVersions enum.

# **Description**

Module providing definitions for the backend types.

## The client.py module

## **Summary**

#### **Classes**

*GrpcClient* Wraps the gRPC connection for the Geometry service.

#### **Functions**

wait\_until\_healthy Wait until a channel is healthy before returning.

# **GrpcClient**

None, backend\_type: an-

| None = None )

sys.geometry.core.connection.backend.BackendType

Wraps the gRPC connection for the Geometry service.

#### **Parameters**

#### host

[str, default: DEFAULT\_HOST] Host where the server is running.

## port

[str or int, default: DEFAULT\_PORT] Port number where the server is running.

#### channel

[Channel, default: None] gRPC channel for server communication.

## remote\_instance

[ansys.platform.instancemanagement.Instance, default: None] Corresponding remote instance when the Geometry service is launched through PyPIM. This instance is deleted when calling the GrpcClient.close method.

## docker\_instance

[LocalDockerInstance, default: None] Corresponding local Docker instance when the Geometry service is launched using the launch\_docker\_modeler() method. This local Docker instance is deleted when the GrpcClient.close method is called.

## product\_instance

[ProductInstance, default: None] Corresponding local product instance when the product (Discovery or SpaceClaim) is

launched through the launch\_modeler\_with\_geometry\_service(), launch\_modeler\_with\_discovery() or the launch\_modeler\_with\_spaceclaim() interface. This instance will be deleted when the GrpcClient.close method is called.

#### timeout

[real, default: 120] Maximum time to spend trying to make the connection.

#### logging level

[int, default: INFO] Logging level to apply to the client.

# logging\_file

[str or Path, default: None] File to output the log to, if requested.

## backend\_type: BackendType, default: None

Type of the backend that PyAnsys Geometry is communicating with. By default, this value is unknown, which results in None being the default value.

## Overview

## **Methods**

close	Close the channel.
target	Get the target of the channel.
get_name	Get the target name of the connection.

# **Properties**

backend_type	Backend type.
backend_version	Get the current backend version.
multiple_designs_allowed	Flag indicating whether multiple designs are allowed.
channel	Client gRPC channel.
log	Specific instance logger.
is_closed	Flag indicating whether the client connection is closed.
healthy	Flag indicating whether the client channel is healthy.

## **Special methods**

## Import detail

from ansys.geometry.core.connection.client import GrpcClient

## **Property detail**

Options are Windows Service, Linux Service, Discovery, and SpaceClaim.

## **Notes**

This method might return None because determining the backend type is not straightforward.

# property GrpcClient.backend\_version: semver.version.Version

Get the current backend version.

#### Returns

#### **Version**

Backend version.

## property GrpcClient.multiple\_designs\_allowed: bool

Flag indicating whether multiple designs are allowed.

## **Notes**

Currently, only one design is allowed per service. This method will always return False.

```
property GrpcClient.channel: grpc.Channel
```

Client gRPC channel.

## property GrpcClient.log: ansys.geometry.core.logger.PyGeometryCustomAdapter

Specific instance logger.

## property GrpcClient.is\_closed: bool

Flag indicating whether the client connection is closed.

## property GrpcClient.healthy: bool

Flag indicating whether the client channel is healthy.

## **Method detail**

```
GrpcClient.__repr__() \rightarrow str
```

Represent the client as a string.

## GrpcClient.close()

Close the channel.

## **Notes**

If an instance of the Geometry service was started using PyPIM, this instance is deleted. Furthermore, if a local Docker instance of the Geometry service was started, it is stopped.

```
GrpcClient.target() \rightarrow str
```

Get the target of the channel.

```
\texttt{GrpcClient.get\_name()} \rightarrow str
```

Get the target name of the connection.

## **Description**

Module providing a wrapped abstraction of the gRPC stubs.

## **Module detail**

client.wait\_until\_healthy(channel: grpc.Channel, timeout: float)

Wait until a channel is healthy before returning.

#### **Parameters**

#### channel

[Channel] Channel that must be established and healthy.

## timeout

[float] Timeout in seconds. Attempts are made with the following backoff strategy:

- Starts with 0.1 seconds.
- If the attempt fails, double the timeout.
- This is repeated until the next timeoff exceeds the value for the remaining time. In that case, a final attempt is made with the remaining time.
- If the total elapsed time exceeds the value for the timeout parameter, a TimeoutError is raised.

#### Raises

#### **TimeoutError**

Raised when the total elapsed time exceeds the value for the timeout parameter.

# The conversions.py module

## **Summary**

## **Functions**

unit_vector_to_grpc_direction	Convert a UnitVector3D class to a unit vector gRPC message.
frame_to_grpc_frame	Convert a Frame class to a frame gRPC message.
plane_to_grpc_plane	Convert a Plane class to a plane gRPC message.
sketch_shapes_to_grpc_geometries	Convert lists of SketchEdge and SketchFace to a gRPC mes-
	sage.
sketch_edges_to_grpc_geometries	Convert a list of SketchEdge to a gRPC message.
sketch_arc_to_grpc_arc	Convert an Arc class to an arc gRPC message.
sketch_ellipse_to_grpc_ellipse	Convert a SketchEllipse class to an ellipse gRPC message.
sketch_circle_to_grpc_circle	Convert a SketchCircle class to a circle gRPC message.
<pre>point3d_to_grpc_point</pre>	Convert a Point3D class to a point gRPC message.
<pre>grpc_point_to_point3d</pre>	Convert a point gRPC message class to a Point3D class.
<pre>point2d_to_grpc_point</pre>	Convert a Point2D class to a point gRPC message.
sketch_polygon_to_grpc_polygon	Convert a Polygon class to a polygon gRPC message.
<pre>sketch_segment_to_grpc_line</pre>	Convert a Segment class to a line gRPC message.
tess_to_pd	Convert an ansys.api.geometry.Tessellation to
	pyvista.PolyData.
<pre>grpc_matrix_to_matrix</pre>	Convert an ansys.api.geometry.Matrix to a Matrix44.
<pre>grpc_frame_to_frame</pre>	Convert a frame gRPC message to a Frame class.
<pre>grpc_surface_to_surface</pre>	Convert a surface gRPC message to a Surface class.
<pre>grpc_curve_to_curve</pre>	Convert a curve gRPC message to a Curve.
curve_to_grpc_curve	Convert a Curve object to a curve gRPC message.
trimmed_curve_to_grpc_trimmed_curve	Convert a TrimmedCurve to a trimmed curve gRPC message.
surface_to_grpc_surface	Convert a Surface object to a surface gRPC message.
<pre>trimmed_surface_to_grpc_trimmed_surfa</pre>	Convert a TrimmedSurface to a trimmed surface gRPC mes-
	sage.
line_to_grpc_line	Convert a Line to a line gRPC message.
<pre>grpc_material_to_material</pre>	Convert a material gRPC message to a Material class.
<pre>grpc_material_property_to_material_pr</pre>	Convert a material property gRPC message to a
	MaterialProperty class.

# **Description**

Module providing for conversions.

## Module detail

 $\label{lem:conversions.unit_vector_to_grpc_direction} (\textit{unit\_vector:} \ ansys.geometry.core.math.vector.UnitVector3D) \\ \rightarrow \ ansys.api.geometry.v0.models\_pb2.Direction$ 

Convert a UnitVector3D class to a unit vector gRPC message.

#### **Parameters**

## unit\_vector

[UnitVector3D] Source vector data.

## Returns

## **GRPCDirection**

Geometry service gRPC direction message.

 $\label{eq:conversions.frame_to_grpc_frame} \textbf{(}\textit{frame:} \ ansys.geometry.core.math.frame.Frame) \rightarrow \\ ansys.api.geometry.v0.models\_pb2.Frame$ 

Convert a Frame class to a frame gRPC message.

# **Parameters** frame [Frame] Source frame data. Returns **GRPCFrame** Geometry service gRPC frame message. The unit for the frame origin is meters. conversions.plane\_to\_grpc\_plane(plane: ansys.geometry.core.math.plane.Plane) $\rightarrow$ ansys.api.geometry.v0.models\_pb2.Plane Convert a Plane class to a plane gRPC message. **Parameters** plane [Plane] Source plane data. **Returns GRPCPlane** Geometry service gRPC plane message. The unit is meters. conversions.sketch\_shapes\_to\_grpc\_geometries(plane: ansys.geometry.core.math.plane.Plane, edges: list/ansys.geometry.core.sketch.edge.SketchEdge], faces: list/ansys.geometry.core.sketch.face.SketchFace/, $only\_one\_curve: bool = False) \rightarrow$ ansys.api.geometry.v0.models\_pb2.Geometries Convert lists of SketchEdge and SketchFace to a gRPC message. **Parameters** plane [Plane] Plane for positioning the 2D sketches. edges [list[SketchEdge]] Source edge data. [list[SketchFace]] Source face data. only\_one\_curve [bool, default: False] Whether to project one curve of the whole set of geometries to enhance performance. Returns **GRPCGeometries** Geometry service gRPC geometries message. The unit is meters. conversions.sketch\_edges\_to\_grpc\_geometries(edges: list/ansys.geometry.core.sketch.edge.SketchEdge/, *plane:* ansys.geometry.core.math.plane.Plane) $\rightarrow$ tuple[list[ansys.api.geometry.v0.models\_pb2.Line], list[ansys.api.geometry.v0.models\_pb2.Arc]] Convert a list of SketchEdge to a gRPC message.

#### **Parameters**

edges

[list[SketchEdge]] Source edge data.

```
plane
                   [Plane] Plane for positioning the 2D sketches.
           Returns
               tuple[list[GRPCLine], list[GRPCArc]]
                   Geometry service gRPC line and arc messages. The unit is meters.
conversions.sketch_arc_to_grpc_arc(arc: ansys.geometry.core.sketch.arc.Arc, plane:
                                           ansys.geometry.core.math.plane.Plane) \rightarrow
                                           ansys.api.geometry.v0.models_pb2.Arc
     Convert an Arc class to an arc gRPC message.
           Parameters
               arc
                   [Arc] Source arc data.
               plane
                   [Plane] Plane for positioning the arc within.
           Returns
               GRPCArc
                   Geometry service gRPC arc message. The unit is meters.
conversions.sketch_ellipse_to_grpc_ellipse(ellipse: ansys.geometry.core.sketch.ellipse.SketchEllipse,
                                                     plane: ansys.geometry.core.math.plane.Plane) \rightarrow
                                                     ansys.api.geometry.v0.models_pb2.Ellipse
     Convert a SketchEllipse class to an ellipse gRPC message.
           Parameters
               ellipse
                   [SketchEllipse] Source ellipse data.
           Returns
               GRPCEllipse
                   Geometry service gRPC ellipse message. The unit is meters.
conversions.sketch_circle_to_grpc_circle(circle: ansys.geometry.core.sketch.circle.SketchCircle, plane:
                                                   ansys.geometry.core.math.plane.Plane) \rightarrow
                                                   ansys.api.geometry.v0.models_pb2.Circle
     Convert a SketchCircle class to a circle gRPC message.
           Parameters
               circle
                   [SketchCircle] Source circle data.
               plane
                   [Plane] Plane for positioning the circle.
           Returns
               GRPCCircle
                   Geometry service gRPC circle message. The unit is meters.
conversions.point3d_to_grpc_point(point: ansys.geometry.core.math.point.Point3D) \rightarrow
                                          ansys.api.geometry.v0.models_pb2.Point
     Convert a Point 3D class to a point gRPC message.
```

```
Parameters
               point
                   [Point3D] Source point data.
           Returns
               GRPCPoint
                   Geometry service gRPC point message. The unit is meters.
conversions.grpc_point_to_point3d(point: ansys.api.geometry.v0.models_pb2.Point) \rightarrow
                                         ansys.geometry.core.math.point.Point3D
     Convert a point gRPC message class to a Point 3D class.
           Parameters
               point
                   [GRPCPoint] Source point data.
           Returns
               Point3D
                   Converted point.
conversions.point2d_to_grpc_point(plane: ansys.geometry.core.math.plane.Plane, point2d:
                                         ansys.geometry.core.math.point.Point2D) \rightarrow
                                         ansys.api.geometry.v0.models_pb2.Point
     Convert a Point2D class to a point gRPC message.
           Parameters
               plane
                   [Plane] Plane for positioning the 2D point.
               point
                   [Point2D] Source point data.
           Returns
               GRPCPoint
                   Geometry service gRPC point message. The unit is meters.
conversions.sketch_polygon_to_grpc_polygon(polygon: ansys.geometry.core.sketch.polygon.Polygon,
                                                    plane: ansys.geometry.core.math.plane.Plane) \rightarrow
                                                    ansys.api.geometry.v0.models_pb2.Polygon
     Convert a Polygon class to a polygon gRPC message.
           Parameters
               polygon
                   [Polygon] Source polygon data.
           Returns
               GRPCPolygon
                   Geometry service gRPC polygon message. The unit is meters.
conversions.sketch_segment_to_grpc_line(segment: ansys.geometry.core.sketch.segment.SketchSegment,
                                                 plane: ansys.geometry.core.math.plane.Plane) \rightarrow
                                                 ansys.api.geometry.v0.models_pb2.Line
     Convert a Segment class to a line gRPC message.
           Parameters
```

#### segment

[SketchSegment] Source segment data.

#### Returns

#### **GRPCLine**

Geometry service gRPC line message. The unit is meters.

conversions.tess\_to\_pd( $tess: ansys.api.geometry.v0.models\_pb2.Tessellation$ )  $\rightarrow$  pyvista.PolyData Convert an ansys.api.geometry.Tessellation to pyvista.PolyData.

conversions.grpc\_matrix\_to\_matrix(m: ansys.api.geometry.v0.models\_pb2.Matrix)  $\rightarrow$  ansys.geometry.core.math.matrix.Matrix44

Convert an ansys.api.geometry.Matrix to a Matrix44.

 $\label{conversions.grpc_frame_to_frame} \textbf{(}\textit{frame: ansys.api.geometry.v0.models\_pb2.Frame)} \rightarrow \textit{ansys.geometry.core.math.frame.Frame}$ 

Convert a frame gRPC message to a Frame class.

## **Parameters**

#### **GRPCFrame**

Geometry service gRPC frame message. The unit for the frame origin is meters.

#### Returns

#### frame

[Frame] Resulting converted frame.

conversions.grpc\_surface\_to\_surface(surface: ansys.api.geometry.v0.models\_pb2.Surface, surface\_type: ansys.geometry.core.designer.face.SurfaceType)  $\rightarrow$  ansys.geometry.core.shapes.surfaces.surface.Surface

Convert a surface gRPC message to a Surface class.

#### **Parameters**

#### surface

[GRPCSurface] Geometry service gRPC surface message.

## Returns

#### Surface

Resulting converted surface.

 $conversions. \textbf{grpc\_curve\_to\_curve}(\textit{curve: ansys.api.geometry.v0.models\_pb2.CurveGeometry}) \rightarrow \textit{ansys.geometry.core.shapes.curve.Curve}$ 

Convert a curve gRPC message to a Curve.

#### **Parameters**

#### curve

[GRPCCurve] Geometry service gRPC curve message.

#### Returns

#### Curve

Resulting converted curve.

 $\label{eq:conversions.curve_to_grpc_curve} corves conversions. \\ \textbf{curve\_to\_grpc\_curve}(\textit{curve}: \textit{ansys.geometry.core.shapes.curves.curve}. \\ \textbf{curve}: \textit{ansys.geometry.core.shapes.curves.curve}. \\ \textbf{curve}: \textit{ansys.geometry.core.shapes.curves.curve}. \\ \textbf{curve}: \textit{ansys.geometry.v0.models\_pb2.CurveGeometry}. \\ \textbf{curve}: \textit{ansys.geometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeometry.v0.models\_pb2.CurveGeome$ 

Convert a Curve object to a curve gRPC message.

#### **Parameters**

```
curve
                   [Curve] Curve to convert.
          Returns
               GRPCCurve
                  Return Curve as a ansys.api.geometry.CurveGeometry message.
conversions.trimmed_curve_to_grpc_trimmed_curve(curve: an-
                                                          sys.geometry.core.shapes.curves.trimmed curve.TrimmedCurve)
                                                          \rightarrow ansys.api.geometry.v0.models_pb2.TrimmedCurve
     Convert a TrimmedCurve to a trimmed curve gRPC message.
          Parameters
               curve
                   [TrimmedCurve] Curve to convert.
          Returns
               GRPCTrimmedCurve
                   Geometry service gRPC TrimmedCurve message.
conversions.surface_to_grpc_surface(surface: ansys.geometry.core.shapes.surfaces.surface) \rightarrow
                                           tuple[ansys.api.geometry.v0.models_pb2.Surface,
                                           ansys.api.geometry.v0.models_pb2.SurfaceType]
     Convert a Surface object to a surface gRPC message.
          Parameters
               surface
                  [Surface] Surface to convert.
          Returns
               GRPCSurface
                   Return Surface as a ansys.api.geometry.Surface message.
               GRPCSurfaceType
                   Return the grpc surface type of Surface.
\verb|conversions.trimmed_surface_to_grpc_trimmed_surface|| \textit{trimmed\_surface}|| \textit{trimmed\_surface}||
                                                               sys.geometry.core.shapes.surfaces.TrimmedSurface)
                                                               ansys.api.geometry.v0.models_pb2.TrimmedSurface
     Convert a TrimmedSurface to a trimmed surface gRPC message.
          Parameters
               trimmed surface
                   [TrimmedSurface] Surface to convert.
          Returns
               GRPCTrimmedSurface
                  Geometry service gRPC TrimmedSurface message.
conversions.line_to_grpc_line(line: ansys.geometry.core.shapes.curves.line.Line) \rightarrow
                                    ansys.api.geometry.v0.models_pb2.Line
     Convert a Line to a line gRPC message.
```

**Parameters** 

line

[Line] Line to convert.

#### Returns

## **GRPCLine**

Geometry service gRPC Line message.

 $\textbf{conversions.grpc\_material\_to\_material}(\textit{material: ansys.api.geometry.v0.models\_pb2.Material}) \rightarrow \textit{ansys.geometry.core.materials.material.Material}$ 

Convert a material gRPC message to a Material class.

#### **Parameters**

#### material

[GRPCMaterial] Material gRPC message.

#### Returns

#### **Material**

Converted material.

conversions.grpc\_material\_property\_to\_material\_property(material\_property: an-

sys.api.geometry.v0.models\_pb2.MaterialProperty)

-

ansys.geometry.core.materials.material.Material Property

Convert a material property gRPC message to a MaterialProperty class.

## **Parameters**

## material\_property

[GRPCMaterialProperty] Material property gRPC message.

#### **Returns**

## MaterialProperty

Converted material property.

## The defaults.py module

## **Summary**

#### **Constants**

DEFAULT_HOST	Default for the HOST name.
DEFAULT_PORT	Default for the HOST port.
MAX_MESSAGE_LENGTH	Default for the gRPC maximum message length.
GEOMETRY_SERVICE_DOCKER_IMAGE	Default for the Geometry service Docker image location.
DEFAULT_PIM_CONFIG	Default for the PIM configuration when running PIM Light.

## **Description**

Module providing default connection parameters.

## Module detail

#### defaults.DEFAULT\_HOST

Default for the HOST name.

By default, PyAnsys Geometry searches for the environment variable ANSRV\_GEO\_HOST, and if this variable does not exist, PyAnsys Geometry uses 127.0.0.1 as the host.

#### defaults.DEFAULT\_PORT: int

Default for the HOST port.

By default, PyAnsys Geometry searches for the environment variable ANSRV\_GEO\_PORT, and if this variable does not exist, PyAnsys Geometry uses 50051 as the port.

## defaults.MAX\_MESSAGE\_LENGTH

Default for the gRPC maximum message length.

By default, PyAnsys Geometry searches for the environment variable PYGEOMETRY\_MAX\_MESSAGE\_LENGTH, and if this variable does not exist, it uses 256Mb as the maximum message length.

## defaults.GEOMETRY\_SERVICE\_DOCKER\_IMAGE = 'ghcr.io/ansys/geometry'

Default for the Geometry service Docker image location.

Tag is dependent on what OS service is requested.

## defaults.DEFAULT\_PIM\_CONFIG

Default for the PIM configuration when running PIM Light.

This parameter is only to be used when PIM Light is being run.

## The docker\_instance.py module

# **Summary**

## **Classes**

LocalDockerInstance	Instantiates a Geometry	y service as a local Docker container.
---------------------	-------------------------	----------------------------------------

# Enums

GeometryContainers	Provides an enum holding the available Geometry services.	

## **Functions**

#### LocalDockerInstance

class ansys.geometry.core.connection.docker\_instance.LocalDockerInstance(port: int =

DEFAULT\_PORT,
connect\_to\_existing\_service:
bool = True,
restart\_if\_existing\_service:
bool = False, name:
str | None = None,
image:
GeometryContainers
| None = None)

Instantiates a Geometry service as a local Docker container.

By default, if a container with the Geometry service already exists at the given port, PyAnsys Geometry connects to it. Otherwise, PyAnsys Geometry tries to launch its own service.

#### **Parameters**

#### port

[int, optional] Localhost port to deploy the Geometry service on or the the Modeler interface to connect to (if it is already deployed). By default, the value is the one for the DEFAULT\_PORT connection parameter.

#### connect\_to\_existing\_service

[bool, default: True] Whether the Modeler interface should connect to a Geometry service already deployed at the specified port.

## restart\_if\_existing\_service

[bool, default: False] Whether the Geometry service (which is already running) should be restarted when attempting connection.

#### name

[str or None, default: None] Name of the Docker container to deploy. The default is None, in which case Docker assigns it a random name.

#### image

[GeometryContainers or None, default: None] The Geometry service Docker image to deploy. The default is None, in which case the LocalDockerInstance class identifies the OS of your Docker engine and deploys the latest version of the Geometry service for that OS.

#### Overview

## **Properties**

container	Docker container object that hosts the deployed Geometry service.
existed_previously	Flag indicating whether the container previously existed.

#### **Attributes**

\_\_DOCKER\_CLIENT\_\_ Docker client class variable.

## Static methods

docker_client	Get the initializedDOCKER_CLIENT object.
is_docker_installed	Check a local installation of Docker engine is available and running.

## Import detail

from ansys.geometry.core.connection.docker\_instance import LocalDockerInstance

# **Property detail**

property LocalDockerInstance.container: docker.models.containers.Container

Docker container object that hosts the deployed Geometry service.

property LocalDockerInstance.existed\_previously: bool

Flag indicating whether the container previously existed.

Returns False if the Geometry service was effectively deployed by this class or True if it already existed.

## **Attribute detail**

LocalDockerInstance.\_\_DOCKER\_CLIENT\_\_: docker.client.DockerClient = None

Docker client class variable.

## **Notes**

The default is None, in which case lazy initialization is used. \_\_DOCKER\_CLIENT\_\_ is a class variable, meaning that it is the same variable for all instances of this class.

## **Method detail**

```
\mbox{\bf static LocalDockerInstance.} \mbox{\bf docker\_client}() \rightarrow \mbox{\bf docker.} \mbox{\bf client.} \mbox{\bf DockerClient} Get the initialized \mbox{\bf \_DOCKER\_CLIENT\_} object.
```

## Returns

DockerClient

Initialized Docker client.

## **Notes**

The LocalDockerInstance class performs a lazy initialization of the \_\_DOCKER\_CLIENT\_\_ class variable.

```
\textbf{static} \ \texttt{LocalDockerInstance}. \textbf{is\_docker\_installed}() \rightarrow bool
```

Check a local installation of Docker engine is available and running.

#### Returns

bool

True if Docker engine is available and running, False otherwise.

#### **GeometryContainers**

Provides an enum holding the available Geometry services.

#### Overview

#### **Attributes**

```
CORE_WINDOWS_LATEST

CORE_LINUX_LATEST

CORE_WINDOWS_LATEST_UNSTABLE

CORE_LINUX_LATEST_UNSTABLE

WINDOWS_LATEST
WINDOWS_LATEST_UNSTABLE

WINDOWS_24_1

WINDOWS_24_2

WINDOWS_25_1

WINDOWS_25_1

WINDOWS_25_2

CORE_WINDOWS_25_2

CORE_LINUX_25_2
```

## Import detail

from ansys.geometry.core.connection.docker\_instance import GeometryContainers

## Attribute detail

```
GeometryContainers.CORE_LINUX_LATEST = (0, 'windows', 'core-windows-latest')

GeometryContainers.CORE_LINUX_LATEST = (1, 'linux', 'core-linux-latest')

GeometryContainers.CORE_WINDOWS_LATEST_UNSTABLE = (2, 'windows', 'core-windows-latest-unstable')

GeometryContainers.CORE_LINUX_LATEST_UNSTABLE = (3, 'linux', 'core-linux-latest-unstable')

GeometryContainers.WINDOWS_LATEST = (4, 'windows', 'windows-latest')

GeometryContainers.WINDOWS_LATEST_UNSTABLE = (5, 'windows', 'windows-latest-unstable')

GeometryContainers.WINDOWS_24_1 = (6, 'windows', 'windows-24.1')

GeometryContainers.WINDOWS_24_2 = (7, 'windows', 'windows-24.2')

GeometryContainers.WINDOWS_25_1 = (8, 'windows', 'windows-25.1')

GeometryContainers.WINDOWS_25_2 = (9, 'windows', 'windows-25.2')

GeometryContainers.CORE_WINDOWS_25_2 = (10, 'windows', 'core-windows-25.2')

GeometryContainers.CORE_LINUX_25_2 = (11, 'linux', 'core-linux-25.2')
```

# **Description**

Module for connecting to a local Geometry Service Docker container.

## Module detail

 $\label{locker_instance} \textbf{docker_instance.get\_geometry\_container\_type} (\textit{instance}: LocalDockerInstance}) \rightarrow \textit{GeometryContainers} \mid \\ \textbf{None}$ 

Provide back the GeometryContainers value.

#### **Parameters**

#### instance

[LocalDockerInstance] The LocalDockerInstance object.

## **Returns**

## GeometryContainers or None

The GeometryContainer value corresponding to the previous image or None if not match.

#### **Notes**

This method returns the first hit on the available tags.

## The launcher.py module

# **Summary**

## **Functions**

launch_modeler	Start the Modeler interface for PyAnsys Geometry.
launch_remote_modeler	Start the Geometry service remotely using the PIM API.
launch_docker_modeler	Start the Geometry service locally using Docker.
<pre>launch_modeler_with_discovery_and_pimlig</pre>	Start Ansys Discovery remotely using the PIM API.
launch_modeler_with_geometry_service_and	Start the Geometry service remotely using the PIM API.
<pre>launch_modeler_with_spaceclaim_and_pimli</pre>	Start Ansys SpaceClaim remotely using the PIM API.
<pre>launch_modeler_with_geometry_service</pre>	Start the Geometry service locally using the ProductInstance class.
launch_modeler_with_discovery	Start Ansys Discovery locally using the ProductInstance class.
launch_modeler_with_spaceclaim	Start Ansys SpaceClaim locally using the ProductInstance class.
launch_modeler_with_core_service	Start the Geometry Core service locally using the ProductInstance class.

## **Description**

Module for connecting to instances of the Geometry service.

## Module detail

launcher.launch\_modeler( $mode: str = None, **kwargs: dict | None) \rightarrow ansys.geometry.core.modeler.Modeler$ Start the Modeler interface for PyAnsys Geometry.

#### **Parameters**

#### mode

[str, default: None] Mode in which to launch the Modeler service. The default is None, in which case the method tries to determine the mode automatically. The possible values are:

- "pypim": Launches the Modeler service remotely using the PIM API.
- "docker": Launches the Modeler service locally using Docker.
- "geometry\_service": Launches the Modeler service locally using the Ansys Geometry Service.
- "spaceclaim": Launches the Modeler service locally using Ansys SpaceClaim.
- "discovery": Launches the Modeler service locally using Ansys Discovery.

#### \*\*kwargs

[dict, default: None] Keyword arguments for the launching methods. For allowable keyword arguments, see the corresponding methods for each mode:

- For "pypim" mode, see the launch\_remote\_modeler() method.
- For "docker" mode, see the launch\_docker\_modeler() method.
- For "geometry\_service" mode, see the launch\_modeler\_with\_geometry\_service() method.
- For "core\_service" mode, see the launch\_modeler\_with\_core\_service() method.
- For "spaceclaim" mode, see the launch\_modeler\_with\_spaceclaim() method.
- For "discovery" mode, see the launch\_modeler\_with\_discovery() method.

#### **Returns**

# ansys.geometry.core.modeler.Modeler

Pythonic interface for geometry modeling.

## **Examples**

Launch the Geometry service.

```
>>> from ansys.geometry.core import launch_modeler
>>> modeler = launch_modeler()
```

```
launcher.launch_remote_modeler(platform: str = 'windows', version: str | None = None, client_log_level: int = logging.INFO, client_log_file: str | None = None, **kwargs: dict | None) \rightarrow ansys.geometry.core.modeler.Modeler
```

Start the Geometry service remotely using the PIM API.

When calling this method, you must ensure that you are in an environment where PyPIM is configured. You can use the pypim.is\_configured method to check if it is configured.

## **Parameters**

## platform

[str, default: None] **Specific for Ansys Lab**. The platform option for the Geometry service. The default is "windows". This parameter is used to specify the operating system on which the Geometry service will run. The possible values are:

- "windows": The Geometry service runs on a Windows machine.
- "linux": The Geometry service runs on a Linux machine.

#### version

[str, default: None] Version of the Geometry service to run in the three-digit format. For example, "241". If you do not specify the version, the server chooses the version.

## client\_log\_level

[int, default: logging.INFO] Log level for the client. The default is logging.INFO.

#### client log file

[str, default: None] Path to the log file for the client. The default is None, in which case the client logs to the console.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

## ansys.geometry.core.modeler.Modeler

Instance of the Geometry service.

```
launcher.launch_docker_modeler(port: int = DEFAULT_PORT, connect_to_existing_service: bool = True, restart_if_existing_service: bool = False, name: str | None = None, image: ansys.geometry.core.connection.docker_instance.GeometryContainers | None = None, client_log_level: int = logging.INFO, client_log_file: str | None = None, **kwargs: dict | None) \rightarrow ansys.geometry.core.modeler.Modeler
```

Start the Geometry service locally using Docker.

When calling this method, a Geometry service (as a local Docker container) is started. By default, if a container with the Geometry service already exists at the given port, it connects to it. Otherwise, it tries to launch its own service.

## **Parameters**

#### port

[int, optional] Localhost port to deploy the Geometry service on or the the Modeler interface to connect to (if it is already deployed). By default, the value is the one for the DEFAULT\_PORT connection parameter.

## connect\_to\_existing\_service

[bool, default: True] Whether the Modeler interface should connect to a Geometry service already deployed at the specified port.

## restart\_if\_existing\_service

[bool, default: False] Whether the Geometry service (which is already running) should be restarted when attempting connection.

#### name

[str, default: None] Name of the Docker container to deploy. The default is None, in which case Docker assigns it a random name.

## image

[GeometryContainers, default: None] The Geometry service Docker image to deploy. The default is None, in which case the LocalDockerInstance class identifies the OS of your Docker engine and deploys the latest version of the Geometry service for that OS.

#### client log level

[int, default: logging.INFO] Log level for the client. The default is logging.INFO.

#### client log file

[str, default: None] Path to the log file for the client. The default is None, in which case the client logs to the console.

#### \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

#### Modeler

Instance of the Geometry service.

```
\label{launcher} \begin{tabular}{ll} launcher. launch_modeler_with_discovery_and_pimlight(\it version: str | None = None, client_log_level: int \\ = logging.INFO, client_log_file: str | None = \\ None, **kwargs: dict | None) \rightarrow \\ ansys. geometry. core. modeler. Modeler \end{tabular}
```

Start Ansys Discovery remotely using the PIM API.

When calling this method, you must ensure that you are in an environment where PyPIM is configured. You can use the pypim.is\_configured method to check if it is configured.

#### **Parameters**

#### version

[str, default: None] Version of Discovery to run in the three-digit format. For example, "241". If you do not specify the version, the server chooses the version.

#### client\_log\_level

[int, default: logging.INFO] Log level for the client. The default is logging.INFO.

#### client log file

[str, default: None] Path to the log file for the client. The default is None, in which case the client logs to the console.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

## ansys.geometry.core.modeler.Modeler

Instance of Modeler.

```
launcher.launch_modeler_with_geometry_service_and_pimlight(version: str \mid None = None, client_log_level: int = logging.INFO, client_log_file: str \mid None = None, **kwargs: dict \mid None) \rightarrow ansys.geometry.core.modeler.Modeler
```

Start the Geometry service remotely using the PIM API.

When calling this method, you must ensure that you are in an environment where PyPIM is configured. You can use the pypim.is\_configured method to check if it is configured.

#### **Parameters**

#### version

[str, default: None] Version of the Geometry service to run in the three-digit format. For example, "241". If you do not specify the version, the server chooses the version.

#### client log level

[int, default: logging.INFO] Log level for the client. The default is logging.INFO.

#### client log file

[str, default: None] Path to the log file for the client. The default is None, in which case the client logs to the console.

#### \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

```
ansys.geometry.core.modeler.Modeler
Instance of Modeler.
```

```
launcher. \textbf{launch\_modeler\_with\_spaceclaim\_and\_pimlight} (\textit{version: str} \mid \textit{None} = \textit{None, client\_log\_level:} \\ int = logging. \textit{INFO, client\_log\_file: str} \mid \textit{None} \\ = \textit{None, **kwargs: dict} \mid \textit{None}) \rightarrow \\ ansys. \textit{geometry.core.modeler.Modeler} \\ \end{cases}
```

Start Ansys SpaceClaim remotely using the PIM API.

When calling this method, you must ensure that you are in an environment where PyPIM is configured. You can use the pypim.is\_configured method to check if it is configured.

#### **Parameters**

#### version

[str, default: None] Version of SpaceClaim to run in the three-digit format. For example, "241". If you do not specify the version, the server chooses the version.

#### client\_log\_level

[int, default: logging.INFO] Log level for the client. The default is logging.INFO.

#### client log file

[str, default: None] Path to the log file for the client. The default is None, in which case the client logs to the console.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

```
ansys.geometry.core.modeler.Modeler
```

Instance of Modeler.

```
launcher.launch_modeler_with_geometry_service(product_version: int = None, host: str = 'localhost', port: int = None, enable_trace: bool = False, timeout: int = 60, server_log_level: int = 2, client_log_level: int = logging.INFO, server_logs_folder: str = None, client_log_file: str = None, logs_folder: str = None, logs_folder: str = None, **kwargs: dict | None) \rightarrow ansys.geometry.core.modeler.Modeler
```

Start the Geometry service locally using the ProductInstance class.

When calling this method, a standalone Geometry service is started. By default, if an endpoint is specified (by defining *host* and *port* parameters) but the endpoint is not available, the startup will fail. Otherwise, it will try to launch its own service.

#### **Parameters**

#### product\_version: int, optional

The product version to be started. Goes from v24.1 to the latest. Default is None. If a specific product version is requested but not installed locally, a SystemError will be raised.

## Ansys products versions and their corresponding int values:

- 241: Ansys 24R1
- 242 : Ansys 24R2

#### host: str, optional

IP address at which the Geometry service will be deployed. By default, its value will be localhost.

#### port

[int, optional] Port at which the Geometry service will be deployed. By default, its value will be None.

## enable\_trace

[bool, optional] Boolean enabling the logs trace on the Geometry service console window. By default its value is False.

#### timeout

[int, optional] Timeout for starting the backend startup process. The default is 60.

## server log level

[int, optional]

## Backend's log level from 0 to 3:

0: Chatterbox 1: Debug 2: Warning 3: Error

The default is 2 (Warning).

## client\_log\_level

[int, optional] Logging level to apply to the client. By default, INFO level is used. Use the logging module's levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

## server\_logs\_folder

[str, optional] Sets the backend's logs folder path. If nothing is defined, the backend will use its default path.

## client\_log\_file

[str, optional] Sets the client's log file path. If nothing is defined, the client will log to the console.

## log level

[int, optional] DEPRECATED. Use server\_log\_level instead.

#### logs folder

[str, optional] DEPRECATED. Use server\_logs\_folder instead.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

## Modeler

Instance of the Geometry service.

## Raises

#### ConnectionError

If the specified endpoint is already in use, a connection error will be raised.

#### SystemError

If there is not an Ansys product 24.1 version or later installed a SystemError will be raised.

# **Examples**

Starting a geometry service with the default parameters and getting back a Modeler object:

```
>>> from ansys.geometry.core import launch_modeler_with_geometry_service
>>> modeler = launch_modeler_with_geometry_service()
```

Starting a geometry service, on address 10.171.22.44, port 5001, with chatty logs, traces enabled and a 300 seconds timeout:

```
launcher.launch_modeler_with_discovery(product_version: int = None, host: str = 'localhost', port: int = None, api_version:

ansys.geometry.core.connection.backend.ApiVersions = ApiVersions.LATEST, timeout: int = 150, manifest_path: str = None, hidden: bool = False, server_log_level: int = 2, client_log_level: int = logging.INFO, client_log_file: str = None, log_level: int = None, **kwargs: dict | None)
```

Start Ansys Discovery locally using the ProductInstance class.

When calling this method, a standalone Discovery session is started. By default, if an endpoint is specified (by defining *host* and *port* parameters) but the endpoint is not available, the startup will fail. Otherwise, it will try to launch its own service.

## **Parameters**

## product\_version: int, optional

The product version to be started. Goes from v24.1 to the latest. Default is None. If a specific product version is requested but not installed locally, a SystemError will be raised.

## Ansys products versions and their corresponding int values:

241 : Ansys 24R1242 : Ansys 24R2

# host: str, optional

IP address at which the Discovery session will be deployed. By default, its value will be localhost.

#### port

[int, optional] Port at which the Geometry service will be deployed. By default, its value will be None.

# api\_version: ApiVersions, optional

The backend's API version to be used at runtime. Goes from API v21 to the latest. Default is ApiVersions.LATEST.

#### timeout

[int, optional] Timeout for starting the backend startup process. The default is 150.

#### manifest path

[str, optional] Used to specify a manifest file path for the ApiServerAddin. This way, it is possible to run an ApiServerAddin from a version an older product version.

#### hidden

[starts the product hiding its UI. Default is False.]

#### server log level

[int, optional]

# Backend's log level from 0 to 3:

0: Chatterbox 1: Debug 2: Warning 3: Error

The default is 2 (Warning).

#### client log level

[int, optional] Logging level to apply to the client. By default, INFO level is used. Use the logging module's levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

## client\_log\_file

[str, optional] Sets the client's log file path. If nothing is defined, the client will log to the console.

#### log level

[int, optional] DEPRECATED. Use server\_log\_level instead.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

#### Modeler

Instance of the Geometry service.

#### Raises

#### ConnectionError

If the specified endpoint is already in use, a connection error will be raised.

#### SystemError

If there is not an Ansys product 24.1 version or later installed or if a specific product's version is requested but not installed locally then a SystemError will be raised.

## **Examples**

Starting an Ansys Discovery session with the default parameters and getting back a Modeler object:

```
>>> from ansys.geometry.core import launch_modeler_with_discovery
>>> modeler = launch_modeler_with_discovery()
```

Starting an Ansys Discovery V 24.1 session, on address 10.171.22.44, port 5001, with chatty logs, using API v231 and a 300 seconds timeout:

```
>>> from ansys.geometry.core import launch_modeler_with_discovery
>>> modeler = launch_modeler_with_discovery(product_version = 241,
    host="10.171.22.44",
    port=5001,
    api_version= 231,
    timeout=300,
    server_log_level=0)
```

launcher.launch\_modeler\_with\_spaceclaim(product\_version: int = None, host: str = 'localhost', port: int = None, api\_version:

ansys.geometry.core.connection.backend.ApiVersions = ApiVersions.LATEST, timeout: int = 150, manifest\_path: str = None, hidden: bool = False, server\_log\_level: int = 2, client\_log\_level: int = logging.INFO, client\_log\_file: str = None, log\_level: int = None, \*\*kwargs: dict | None)

Start Ansys SpaceClaim locally using the ProductInstance class.

When calling this method, a standalone SpaceClaim session is started. By default, if an endpoint is specified (by defining *host* and *port* parameters) but the endpoint is not available, the startup will fail. Otherwise, it will try to launch its own service.

#### **Parameters**

#### product\_version: int, optional

The product version to be started. Goes from v24.1 to the latest. Default is None. If a specific product version is requested but not installed locally, a SystemError will be raised.

## Ansys products versions and their corresponding int values:

- 241 : Ansys 24R1
- 242 : Ansys 24R2

## host: str, optional

IP address at which the SpaceClaim session will be deployed. By default, its value will be localhost.

#### port

[int, optional] Port at which the Geometry service will be deployed. By default, its value will be None.

## api\_version: ApiVersions, optional

The backend's API version to be used at runtime. Goes from API v21 to the latest. Default is ApiVersions.LATEST.

#### timeout

[int, optional] Timeout for starting the backend startup process. The default is 150.

#### manifest\_path

[str, optional] Used to specify a manifest file path for the ApiServerAddin. This way, it is possible to run an ApiServerAddin from a version an older product version.

#### hidden

[starts the product hiding its UI. Default is False.]

#### server\_log\_level

[int, optional]

#### Backend's log level from 0 to 3:

0: Chatterbox 1: Debug 2: Warning 3: Error

The default is 2 (Warning).

#### client log level

[int, optional] Logging level to apply to the client. By default, INFO level is used. Use the logging module's levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

## client\_log\_file

[str, optional] Sets the client's log file path. If nothing is defined, the client will log to the console.

#### log level

[int, optional] DEPRECATED. Use server\_log\_level instead.

## \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

#### Returns

#### Modeler

Instance of the Geometry service.

#### Raises

#### ConnectionError

If the specified endpoint is already in use, a connection error will be raised.

#### SystemError

If there is not an Ansys product 24.1 version or later installed or if a specific product's version is requested but not installed locally then a SystemError will be raised.

## **Examples**

Starting an Ansys SpaceClaim session with the default parameters and get back a Modeler object:

```
>>> from ansys.geometry.core import launch_modeler_with_spaceclaim
>>> modeler = launch_modeler_with_spaceclaim()
```

Starting an Ansys SpaceClaim V 24.1 session, on address 10.171.22.44, port 5001, with chatty logs, using API v231 and a 300 seconds timeout:

```
>>> from ansys.geometry.core import launch_modeler_with_spaceclaim
>>> modeler = launch_modeler_with_spaceclaim(product_version = 241,
    host="10.171.22.44",
    port=5001,
    api_version= 231,
    timeout=300,
    server_log_level=0)
```

```
launcher.launch_modeler_with_core_service(product_version: int = None, host: str = 'localhost', port: int = None, enable_trace: bool = False, timeout: int = 60, server_log_level: int = 2, client_log_level: int = logging.INFO, server_logs_folder: <math>str = None, client_log_file: str = None, **kwargs: dict | None) \rightarrow ansys.geometry.core.modeler.Modeler
```

Start the Geometry Core service locally using the ProductInstance class.

When calling this method, a standalone Geometry Core service is started. By default, if an endpoint is specified (by defining *host* and *port* parameters) but the endpoint is not available, the startup will fail. Otherwise, it will try to launch its own service.

## **Parameters**

## product\_version: int, optional

The product version to be started. Goes from v25.2 to the latest. Default is None. If a specific product version is requested but not installed locally, a SystemError will be raised.

Ansys products versions and their corresponding int values:

• 252 : Ansys 25R2

• 261 : Ansys 26R1

#### host: str, optional

IP address at which the service will be deployed. By default, its value will be localhost.

#### port

[int, optional] Port at which the service will be deployed. By default, its value will be None.

#### enable trace

[bool, optional] Boolean enabling the logs trace on the service console window. By default its value is False.

#### timeout

[int, optional] Timeout for starting the backend startup process. The default is 60.

#### server log level

[int, optional]

## Backend's log level from 0 to 3:

0: Chatterbox 1: Debug 2: Warning 3: Error

The default is 2 (Warning).

## client\_log\_level

[int, optional] Logging level to apply to the client. By default, INFO level is used. Use the logging module's levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

#### server logs folder

[str, optional] Sets the backend's logs folder path. If nothing is defined, the backend will use its default path.

## client\_log\_file

[str, optional] Sets the client's log file path. If nothing is defined, the client will log to the console.

#### \*\*kwargs

[dict, default: None] Placeholder to prevent errors when passing additional arguments that are not compatible with this method.

## Returns

#### Modeler

Instance of the Geometry Core service.

#### Raises

#### ConnectionError

If the specified endpoint is already in use, a connection error will be raised.

## **SystemError**

If there is not an Ansys product 25.2 version or later installed a SystemError will be raised.

## **Examples**

Starting a geometry core service with the default parameters and getting back a Modeler object:

```
>>> from ansys.geometry.core import launch_modeler_with_core_service
>>> modeler = launch_modeler_with_core_service()
```

Starting a geometry service, on address 10.171.22.44, port 5001, with chatty logs, traces enabled and a 300 seconds timeout:

# The product\_instance.py module

# **Summary**

# Classes

ProductInstance ProductInstance class.

## **Functions**

prepare_and_start_backend	Start the requested service locally using the ProductInstance class.
get_available_port	Return an available port to be used.

## **Constants**

WINDOWS_GEOMETRY_SERVICE_FOLDE	Default Geometry Service's folder name into the unified installer (DMS).
CORE_GEOMETRY_SERVICE_FOLDER	Default Geometry Service's folder name into the unified installer (Core Service).
DISCOVERY_FOLDER	Default Discovery's folder name into the unified installer.
SPACECLAIM_FOLDER	Default SpaceClaim's folder name into the unified installer.
ADDINS_SUBFOLDER	Default global Addins's folder name into the unified installer.
BACKEND_SUBFOLDER	Default backend's folder name into the ADDINS_SUBFOLDER folder.
MANIFEST_FILENAME	Default backend's add-in filename.
GEOMETRY_SERVICE_EXE	The Windows Geometry Service's filename (DMS).
CORE_GEOMETRY_SERVICE_EXE	The Windows Geometry Service's filename (Core Service).
DISCOVERY_EXE	The Ansys Discovery's filename.
SPACECLAIM_EXE	The Ansys SpaceClaim's filename.
BACKEND_LOG_LEVEL_VARIABLE	The backend's log level environment variable for local start.
BACKEND_TRACE_VARIABLE	The backend's enable trace environment variable for local start.
BACKEND_HOST_VARIABLE	The backend's ip address environment variable for local start.
BACKEND_PORT_VARIABLE	The backend's port number environment variable for local start.
BACKEND_LOGS_FOLDER_VARIABLE	The backend's logs folder path to be used.
BACKEND_API_VERSION_VARIABLE	The backend's api version environment variable for local start.
BACKEND_SPACECLAIM_OPTIONS	The additional argument for local Ansys Discovery start.
BACKEND_ADDIN_MANIFEST_ARGUMEN	The argument to specify the backend's add-in manifest file's path.
BACKEND_SPACECLAIM_HIDDEN	The argument to hide SpaceClaim's UI on the backend.
BACKEND_SPACECLAIM_HIDDEN_ENVV	SpaceClaim hidden backend's environment variable key.
BACKEND_SPACECLAIM_HIDDEN_ENVV	SpaceClaim hidden backend's environment variable value.
BACKEND_DISCOVERY_HIDDEN	The argument to hide Discovery's UI on the backend.
BACKEND_SPLASH_OFF	The argument to specify the backend's add-in manifest file's path.
ANSYS_GEOMETRY_SERVICE_ROOT	Local Geometry Service install location. This is for Geometry Service and
	CoreGeometryService.

#### **ProductInstance**

class ansys.geometry.core.connection.product\_instance.ProductInstance(pid: int)

ProductInstance class.

This class is used as a handle for a local session of Ansys Product's backend: Discovery, Windows Geometry Service or SpaceClaim.

#### **Parameters**

pid

[int] The local instance's process identifier. This allows to keep track of the process and close it if need be.

#### Overview

## **Methods**

*close* Close the process associated to the pid.

# Import detail

from ansys.geometry.core.connection.product\_instance import ProductInstance

#### Method detail

ProductInstance.close()  $\rightarrow$  bool

Close the process associated to the pid.

## **Description**

Module containing the ProductInstance class.

#### Module detail

product\_instance.prepare\_and\_start\_backend(backend\_type:

```
ansys.geometry.core.connection.backend.BackendType, product_version: int = None, host: str = 'localhost', port: int = None, enable_trace: bool = False, api_version: ansys.geometry.core.connection.backend.ApiVersions = ApiVersions.LATEST, timeout: int = 150, manifest_path: str = None, hidden: bool = False, server_log_level: int = 2, client_log_level: int = logging.INFO, server_logs_folder: str = None, client_log_file: str = None, specific_minimum_version: int = None, log_level: int = None, logs_folder: str = None) → ansys.geometry.core.modeler.Modeler
```

Start the requested service locally using the ProductInstance class.

When calling this method, a standalone service or product session is started. By default, if an endpoint is specified (by defining *host* and *port* parameters) but the endpoint is not available, the startup will fail. Otherwise, it will try to launch its own service.

#### **Parameters**

## product\_version: ``int``, optional

The product version to be started. Goes from v24.1 to the latest. Default is None. If a specific product version is requested but not installed locally, a SystemError will be raised.

## host: str, optional

IP address at which the Geometry service will be deployed. By default, its value will be localhost.

#### port

[int, optional] Port at which the Geometry service will be deployed. By default, its value will be None.

#### enable trace

[bool, optional] Boolean enabling the logs trace on the Geometry service console window. By default its value is False.

# api\_version: ``ApiVersions``, optional

The backend's API version to be used at runtime. Goes from API v21 to the latest. Default is ApiVersions.LATEST.

#### timeout

[int, optional] Timeout for starting the backend startup process. The default is 150.

## manifest\_path

[str, optional] Used to specify a manifest file path for the ApiServerAddin. This way, it is possible to run an ApiServerAddin from a version an older product version. Only applicable for Ansys Discovery and Ansys SpaceClaim.

#### hidden

[starts the product hiding its UI. Default is False.]

## server\_log\_level

[int, optional]

## Backend's log level from 0 to 3:

0: Chatterbox 1: Debug 2: Warning 3: Error

The default is 2 (Warning).

#### client\_log\_level

[int, optional] Logging level to apply to the client. By default, INFO level is used. Use the logging module's levels: DEBUG, INFO, WARNING, ERROR, CRITICAL.

# server\_logs\_folder

[str, optional] Sets the backend's logs folder path. If nothing is defined, the backend will use its default path.

#### client log file

[str, optional] Sets the client's log file path. If nothing is defined, the client will log to the console.

# specific\_minimum\_version

[int, optional] Sets a specific minimum version to be checked. If this is not defined, the minimum version will be set to 24.1.0.

#### log level

[int, optional] DEPRECATED. Use server\_log\_level instead.

## logs\_folder

[str, optional] DEPRECATED. Use server\_logs\_folder instead.

#### Returns

#### Modeler

Instance of the Geometry service.

#### Raises

#### ConnectionError

If the specified endpoint is already in use, a connection error will be raised.

## SystemError

If there is not an Ansys product 24.1 version or later installed or if a specific product's version is requested but not installed locally then a SystemError will be raised.

## $product\_instance.get\_available\_port() \rightarrow int$

Return an available port to be used.

## **Returns**

int

The available port.

product\_instance.WINDOWS\_GEOMETRY\_SERVICE\_FOLDER = 'GeometryService'

Default Geometry Service's folder name into the unified installer (DMS).

product\_instance.CORE\_GEOMETRY\_SERVICE\_FOLDER = 'CoreGeometryService'

Default Geometry Service's folder name into the unified installer (Core Service).

product\_instance.DISCOVERY\_FOLDER = 'Discovery'

Default Discovery's folder name into the unified installer.

product\_instance.SPACECLAIM\_FOLDER = 'scdm'

Default SpaceClaim's folder name into the unified installer.

product\_instance.ADDINS\_SUBFOLDER = 'Addins'

Default global Addins's folder name into the unified installer.

product\_instance.BACKEND\_SUBFOLDER = 'ApiServer'

Default backend's folder name into the ADDINS\_SUBFOLDER folder.

product\_instance.MANIFEST\_FILENAME = 'Presentation.ApiServerAddIn.Manifest.xml'

Default backend's add-in filename.

To be used only for local start of Ansys Discovery or Ansys SpaceClaim.

product\_instance.GEOMETRY\_SERVICE\_EXE = 'Presentation.ApiServerDMS.exe'

The Windows Geometry Service's filename (DMS).

product\_instance.CORE\_GEOMETRY\_SERVICE\_EXE = 'Presentation.ApiServerLinux.exe'

The Windows Geometry Service's filename (Core Service).

product\_instance.DISCOVERY\_EXE = 'Discovery.exe'

The Ansys Discovery's filename.

product\_instance.SPACECLAIM\_EXE = 'SpaceClaim.exe'

The Ansys SpaceClaim's filename.

product\_instance.BACKEND\_LOG\_LEVEL\_VARIABLE = 'LOG\_LEVEL'

The backend's log level environment variable for local start.

product\_instance.BACKEND\_TRACE\_VARIABLE = 'ENABLE\_TRACE'

The backend's enable trace environment variable for local start.

#### product\_instance.BACKEND\_HOST\_VARIABLE = 'API\_ADDRESS'

The backend's ip address environment variable for local start.

## product\_instance.BACKEND\_PORT\_VARIABLE = 'API\_PORT'

The backend's port number environment variable for local start.

## product\_instance.BACKEND\_LOGS\_FOLDER\_VARIABLE = 'ANS\_DSCO\_REMOTE\_LOGS\_FOLDER'

The backend's logs folder path to be used.

Only applicable to the Ansys Geometry Service.

## product\_instance.BACKEND\_API\_VERSION\_VARIABLE = 'API\_VERSION'

The backend's api version environment variable for local start.

To be used only with Ansys Discovery and Ansys SpaceClaim.

## product\_instance.BACKEND\_SPACECLAIM\_OPTIONS = '--spaceclaim-options'

The additional argument for local Ansys Discovery start.

To be used only with Ansys Discovery.

## product\_instance.BACKEND\_ADDIN\_MANIFEST\_ARGUMENT = '/ADDINMANIFESTFILE='

The argument to specify the backend's add-in manifest file's path.

To be used only with Ansys Discovery and Ansys SpaceClaim.

# product\_instance.BACKEND\_SPACECLAIM\_HIDDEN = '/Headless=True'

The argument to hide SpaceClaim's UI on the backend.

To be used only with Ansys SpaceClaim.

## product\_instance.BACKEND\_SPACECLAIM\_HIDDEN\_ENVVAR\_KEY = 'SPACECLAIM\_MODE'

SpaceClaim hidden backend's environment variable key.

To be used only with Ansys SpaceClaim.

## product\_instance.BACKEND\_SPACECLAIM\_HIDDEN\_ENVVAR\_VALUE = '2'

SpaceClaim hidden backend's environment variable value.

To be used only with Ansys SpaceClaim.

## product\_instance.BACKEND\_DISCOVERY\_HIDDEN = '--hidden'

The argument to hide Discovery's UI on the backend.

To be used only with Ansys Discovery.

#### product\_instance.BACKEND\_SPLASH\_OFF = '/Splash=False'

The argument to specify the backend's add-in manifest file's path.

To be used only with Ansys Discovery and Ansys SpaceClaim.

## product\_instance.ANSYS\_GEOMETRY\_SERVICE\_ROOT = 'ANSYS\_GEOMETRY\_SERVICE\_ROOT'

Local Geometry Service install location. This is for GeometryService and CoreGeometryService.

## The validate.py module

## **Summary**

## **Functions**

validate Create a client using the default settings and validate it.

# **Description**

Module to perform a connection validation check.

The method in this module is only used for testing the default Docker service on GitHub and can safely be skipped within testing.

This command shows how this method is typically used:

```
python -c "from ansys.geometry.core.connection import validate; validate()"
```

## Module detail

validate.validate(\*args, \*\*kwargs)

Create a client using the default settings and validate it.

# **Description**

PyAnsys Geometry connection subpackage.

## The designer package

## **Summary**

## **Submodules**

beam	Provides for creating and managing a beam.
body	Provides for managing a body.
component	Provides for managing components.
<pre>coordinate_system</pre>	Provides for managing a user-defined coordinate system.
design	Provides for managing designs.
designpoint	Module for creating and managing design points.
edge	Module for managing an edge.
face	Module for managing a face.
<pre>geometry_commands</pre>	Provides tools for pulling geometry.
part	Module providing fundamental data of an assembly.
selection	Module for creating a named selection.

## The beam.py module

## **Summary**

#### **Classes**

BeamProfile	Represents a single beam profile organized within the design assembly.
BeamCircularProfile	Represents a single circular beam profile.
Beam	Represents a simplified solid body with an assigned 2D cross-section.

#### **BeamProfile**

class ansys.geometry.core.designer.beam.BeamProfile(id: str, name: str)

Represents a single beam profile organized within the design assembly.

This profile synchronizes to a design within a supporting Geometry service instance.

#### **Parameters**

id

[str] Server-defined ID for the beam profile.

#### name

[str] User-defined label for the beam profile.

#### **Notes**

BeamProfile objects are expected to be created from the Design object. This means that you are not expected to instantiate your own BeamProfile object. You should call the specific Design API for the BeamProfile desired.

## **Overview**

# **Properties**

id	ID of the beam profile.
name	Name of the beam profile.

## Import detail

```
from ansys.geometry.core.designer.beam import BeamProfile
```

## **Property detail**

## BeamCircularProfile

class ansys.geometry.core.designer.beam.BeamCircularProfile(id: str, name: str, radius: an-

sys.geometry.core.misc.measurements. Distance,

center: an-

sys.geometry.core.math.point.Point3D,

direction x: an-

sys.geometry.core.math.vector.UnitVector3D,

direction\_y: an-

sys.geometry.core.math.vector.UnitVector3D)

Bases: BeamProfile

Represents a single circular beam profile.

This profile synchronizes to a design within a supporting Geometry service instance.

#### **Parameters**

id

[str] Server-defined ID for the beam profile.

#### name

[str] User-defined label for the beam profile.

#### radius

[Distance] Radius of the circle.

#### center: Point3D

3D point representing the center of the circle.

## direction x: UnitVector3D

X-axis direction.

## direction y: UnitVector3D

Y-axis direction.

#### **Notes**

BeamProfile objects are expected to be created from the Design object. This means that you are not expected to instantiate your own BeamProfile object. You should call the specific Design API for the BeamProfile desired.

#### Overview

## **Properties**

radius	Radius of the circular beam profile.
center	Center of the circular beam profile.
direction_x	X-axis direction of the circular beam profile.
direction_y	Y-axis direction of the circular beam profile.

# **Special methods**

repr	Represent the BeamCircularProfile as a string.
------	------------------------------------------------

# Import detail

from ansys.geometry.core.designer.beam import BeamCircularProfile

## **Property detail**

property BeamCircularProfile.radius: ansys.geometry.core.misc.measurements.Distance
 Radius of the circular beam profile.

property BeamCircularProfile.direction\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-axis direction of the circular beam profile.

property BeamCircularProfile.direction\_y: ansys.geometry.core.math.vector.UnitVector3D
 Y-axis direction of the circular beam profile.

## **Method detail**

 $\texttt{BeamCircularProfile.\_\_repr}\_\_() \to str$ 

Represent the BeamCircularProfile as a string.

#### **Beam**

Represents a simplified solid body with an assigned 2D cross-section.

This body synchronizes to a design within a supporting Geometry service instance.

#### **Parameters**

id

[str] Server-defined ID for the body.

name

[str] User-defined label for the body.

start

[Point3D] Start of the beam line segment.

end

[Point3D] End of the beam line segment.

profile

[BeamProfile] Beam profile to use to create the beam.

## parent\_component

[Component] Parent component to nest the new beam under within the design assembly.

## **Overview**

# **Properties**

id	Service-defined ID of the beam.
start	Start of the beam line segment.
end	End of the beam line segment.
profile	Beam profile of the beam line segment.
<pre>parent_component</pre>	Component node that the beam is under.
is_alive	Flag indicating whether the beam is still alive on the server.

## **Special methods**

\_\_repr\_\_ Represent the beam as a string.

## Import detail

from ansys.geometry.core.designer.beam import Beam

# **Property detail**

property Beam.id: str

Service-defined ID of the beam.

property Beam.start: ansys.geometry.core.math.point.Point3D

Start of the beam line segment.

property Beam.end: ansys.geometry.core.math.point.Point3D

End of the beam line segment.

property Beam.profile: BeamProfile

Beam profile of the beam line segment.

property Beam.parent\_component: ansys.geometry.core.designer.component.Component

Component node that the beam is under.

property Beam.is\_alive: bool

Flag indicating whether the beam is still alive on the server.

### **Method detail**

Beam.\_\_repr\_\_()  $\rightarrow$  str

Represent the beam as a string.

# **Description**

Provides for creating and managing a beam.

# The body.py module

# Summary

### Interfaces

*IBody* Defines the common methods for a body, providing the abstract body interface.

### Classes

MasterBody	Represents solids and surfaces organized within the design assembly.
Body	Represents solids and surfaces organized within the design assembly.

### **Enums**

MidSurfaceOffsetType	Provides values for mid-surface offsets supported.
CollisionType	Provides values for collision types between bodies.
FillStyle	Provides values for fill styles supported.

# **Constants**

\_TEMPORARY\_BOOL\_OPS\_FIX\_\_

# **IBody**

class ansys.geometry.core.designer.body.IBody

Bases: abc.ABC

Defines the common methods for a body, providing the abstract body interface.

Both the MasterBody class and Body class both inherit from the IBody class. All child classes must implement all abstract methods.

# **Overview**

# **Abstract methods**

id	Get the ID of the body as a string.
name	Get the name of the body.
set_name	Set the name of the body.
fill_style	Get the fill style of the body.
set_fill_style	Set the fill style of the body.
is_suppressed	Get the body suppression state.
set_suppressed	Set the body suppression state.
color	Get the color of the body.
set_color	Set the color of the body.
faces	Get a list of all faces within the body.
edges	Get a list of all edges within the body.
is_alive	Check if the body is still alive and has not been deleted.
is_surface	Check if the body is a planar body.
surface_thickness	Get the surface thickness of a surface body.
surface_offset	Get the surface offset type of a surface body.
volume	Calculate the volume of the body.
material	Get the assigned material of the body.
assign_material	Assign a material against the active design.
<pre>get_assigned_material</pre>	Get the assigned material of the body.
add_midsurface_thickness	Add a mid-surface thickness to a surface body.
add_midsurface_offset	Add a mid-surface offset to a surface body.
imprint_curves	Imprint all specified geometries onto specified faces of the body.
project_curves	Project all specified geometries onto the body.
<pre>imprint_projected_curves</pre>	Project and imprint specified geometries onto the body.
translate	Translate the body in a specified direction and distance.
rotate	Rotate the geometry body around the specified axis by a given angle.
scale	Scale the geometry body by the given value.
map	Map the geometry body to the new specified frame.
mirror	Mirror the geometry body across the specified plane.
<pre>get_collision</pre>	Get the collision state between bodies.
copy	Create a copy of the body under the specified parent.
tessellate	Tessellate the body and return the geometry as triangles.
shell_body	Shell the body to the thickness specified.
remove_faces	Shell by removing a given set of faces.

continues on next page

# Table 1 – continued from previous page

plot	Plot the body.	
------	----------------	--

#### Methods

intersect	Intersect two (or more) bodies.
subtract	Subtract two (or more) bodies.
unite	Unite two (or more) bodies.

# Import detail

```
from ansys.geometry.core.designer.body import IBody
```

### Method detail

```
abstractmethod IBody.id() \rightarrow str
      Get the ID of the body as a string.
abstractmethod IBody.name() \rightarrow str
      Get the name of the body.
abstractmethod IBody.set_name(str) \rightarrow None
      Set the name of the body.
abstractmethod IBody.fill_style() \rightarrow FillStyle
      Get the fill style of the body.
abstractmethod IBody.set_fill_style(fill_style: FillStyle) \rightarrow None
      Set the fill style of the body.
abstractmethod IBody.is_suppressed() → bool
      Get the body suppression state.
abstractmethod IBody.set\_suppressed(suppressed: bool) \rightarrow None
      Set the body suppression state.
abstractmethod IBody.color() \rightarrow str
      Get the color of the body.
abstractmethod IBody.set_color(color: str \mid tuple[float, float]) \rightarrow None
      Set the color of the body.
           Parameters
                color
                    [str | tuple[float, float, float]] Color to set the body to. This can be a string repre-
                    senting a color name or a tuple of RGB values in the range [0, 1] (RGBA) or [0, 255] (pure
                    RGB).
abstractmethod IBody.faces() \rightarrow list[ansys.geometry.core.designer.face.Face]
      Get a list of all faces within the body.
           Returns
                list[Face]
```

### **abstractmethod** IBody.edges() $\rightarrow$ list[ansys.geometry.core.designer.edge.Edge]

Get a list of all edges within the body.

### Returns

### list[Edge]

### abstractmethod IBody.is\_alive() $\rightarrow$ bool

Check if the body is still alive and has not been deleted.

# $abstractmethod IBody.is\_surface() \rightarrow bool$

Check if the body is a planar body.

### **abstractmethod** IBody.**surface\_thickness()** → pint.Quantity | None

Get the surface thickness of a surface body.

### **Notes**

This method is only for surface-type bodies that have been assigned a surface thickness.

# $abstractmethod IBody.surface\_offset() \rightarrow MidSurfaceOffsetType \mid None$

Get the surface offset type of a surface body.

### **Notes**

This method is only for surface-type bodies that have been assigned a surface offset.

# $abstractmethod IBody.volume() \rightarrow pint.Quantity$

Calculate the volume of the body.

# **Notes**

When dealing with a planar surface, a value of 0 is returned as a volume.

# $abstractmethod \ IBody.material() \rightarrow ansys.geometry.core.materials.material.Material$

Get the assigned material of the body.

### Returns

### **Material**

Material assigned to the body.

**abstractmethod** IBody.assign\_material(material: ansys.geometry.core.materials.material.Material)  $\rightarrow$  None Assign a material against the active design.

#### **Parameters**

### material

[Material] Source material data.

# $\textbf{abstractmethod} \hspace{0.2cm} \textbf{IBody.get\_assigned\_material()} \rightarrow \textit{ansys.geometry.core.materials.material.Material}$

Get the assigned material of the body.

### Returns

### **Material**

Material assigned to the body.

# $abstractmethod\ IBody.add\_midsurface\_thickness:\ pint.Quantity) \rightarrow None$

Add a mid-surface thickness to a surface body.

### **Parameters**

#### thickness

[Quantity] Thickness to assign.

### **Notes**

Only surface bodies are eligible for mid-surface thickness assignment.

# **abstractmethod** IBody.add\_midsurface\_offset(offset: MidSurfaceOffsetType) → None

Add a mid-surface offset to a surface body.

#### **Parameters**

# offset\_type

[MidSurfaceOffsetType] Surface offset to assign.

### **Notes**

Only surface bodies are eligible for mid-surface offset assignment.

```
abstractmethod IBody.imprint_curves(faces: list[ansys.geometry.core.designer.face.Face], sketch:
```

ansys.geometry.core.sketch.sketch.Sketch) → tuple[list[ansys.geometry.core.designer.edge.Edge], list[ansys.geometry.core.designer.face.Face]]

Imprint all specified geometries onto specified faces of the body.

### **Parameters**

### faces: list[Face]

list of faces to imprint the curves of the sketch onto.

### sketch: Sketch

All curves to imprint on the faces.

### Returns

```
tuple[list[Edge], list[Face]]
```

All impacted edges and faces from the imprint operation.

### **abstractmethod** IBody.**project\_curves**(*direction:* ansys.geometry.core.math.vector.UnitVector3D, *sketch:*

ansys.geometry.core.sketch.sketch.Sketch,  $closest\_face: bool$ ,  $only\_one\_curve: bool = False$ )  $\rightarrow$ 

list[ansys.geometry.core.designer.face.Face]

Project all specified geometries onto the body.

### **Parameters**

# direction: UnitVector3D

Direction of the projection.

#### sketch: Sketch

All curves to project on the body.

### closest\_face: bool

Whether to target the closest face with the projection.

#### only one curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### Returns

### list[Face]

All faces from the project curves operation.

### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

### abstractmethod IBody.imprint\_projected\_curves(direction:

ansys.geometry.core.math.vector.UnitVector3D, *sketch*: ansys.geometry.core.sketch.sketch.Sketch, *closest\_face*: bool,  $only\_one\_curve$ : bool = False)  $\rightarrow$  list[ansys.geometry.core.designer.face.Face]

Project and imprint specified geometries onto the body.

This method combines the project\_curves() and imprint\_curves() method into one method. It has higher performance than calling them back-to-back when dealing with many curves. Because it is a specialized function, this method only returns the faces (and not the edges) from the imprint operation.

### **Parameters**

### direction: UnitVector3D

Direction of the projection.

### sketch: Sketch

All curves to project on the body.

### closest\_face: bool

Whether to target the closest face with the projection.

### only\_one\_curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### Returns

# list[Face]

All imprinted faces from the operation.

### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

**abstractmethod** IBody.**translate**(direction: ansys.geometry.core.math.vector.UnitVector3D, distance:  $pint.Quantity \mid ansys.geometry.core.misc.measurements.Distance \mid ansys.geometry.core.typing.Real) <math>\rightarrow$  None

Translate the body in a specified direction and distance.

#### **Parameters**

### direction: UnitVector3D

Direction of the translation.

# distance: ~pint.Quantity | Distance | Real

Distance (magnitude) of the translation.

### Returns

None

```
abstractmethod IBody.rotate(axis\_origin: ansys.geometry.core.math.point.Point3D, axis\_direction: ansys.geometry.core.math.vector.UnitVector3D, angle: pint.Quantity \mid ansys.geometry.core.misc.measurements.Angle \mid ansys.geometry.core.typing.Real) \rightarrow None
```

Rotate the geometry body around the specified axis by a given angle.

#### **Parameters**

axis\_origin: Point3D
Origin of the rotational axis.

axis\_direction: UnitVector3D
The axis of rotation.

angle: ~pint.Quantity | Angle | Real Angle (magnitude) of the rotation.

### Returns

None

**abstractmethod** IBody.**scale**( $value: ansys.geometry.core.typing.Real) \rightarrow None Scale the geometry body by the given value.$ 

#### **Parameters**

value: Real

Value to scale the body by.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

 $\textbf{abstractmethod} \hspace{0.2cm} \textbf{IBody.map}(\textit{frame:} \hspace{0.2cm} ansys.geometry.core.math.frame.Frame) \rightarrow None$ 

Map the geometry body to the new specified frame.

### **Parameters**

### frame: Frame

Structure defining the orientation of the body.

# **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

**abstractmethod** IBody.mirror(plane: ansys.geometry.core.math.plane.Plane)  $\rightarrow$  None

Mirror the geometry body across the specified plane.

### **Parameters**

### plane: Plane

Represents the mirror.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

### **abstractmethod** IBody.get\_collision(body: Body) $\rightarrow$ CollisionType

Get the collision state between bodies.

### **Parameters**

### body: Body

Object that the collision state is checked with.

### Returns

### CollisionType

Enum that defines the collision state between bodies.

**abstractmethod** IBody.**copy**(parent: ansys.geometry.core.designer.component.Component, name: str = None)  $\rightarrow Body$ 

Create a copy of the body under the specified parent.

### **Parameters**

# parent: Component

Parent component to place the new body under within the design assembly.

#### name: str

Name to give the new body.

### Returns

#### **Body**

Copy of the body.

**abstractmethod** IBody.tessellate(merge: bool = False)  $\rightarrow$  pyvista.PolyData|pyvista.MultiBlock

Tessellate the body and return the geometry as triangles.

### **Parameters**

### merge

[bool, default: False] Whether to merge the body into a single mesh. When False (default), the number of triangles are preserved and only the topology is merged. When True, the individual faces of the tessellation are merged.

### Returns

### PolyData, MultiBlock

Merged pyvista.PolyData if merge=True or a composite dataset.

### **Examples**

Extrude a box centered at the origin to create a rectangular body and tessellate it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> my_comp = design.add_component("my-comp")
```

(continues on next page)

(continued from previous page)

```
>>> body = my_comp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> blocks = body.tessellate()
>>> blocks
>>> MultiBlock(0x7F94EC757460)

    N Blocks: 6
    X Bounds: 0.000, 4.000
    Y Bounds: -1.000, 0.000
    Z Bounds: -0.500, 4.500
```

Merge the body:

abstractmethod IBody. $shell\_body(\mathit{offset: ansys.geometry.core.typing.Real}) \rightarrow bool$ 

Shell the body to the thickness specified.

### **Parameters**

offset

[Real] Shell thickness.

### Returns

bool

True when successful, False when failed.

```
\begin{tabular}{ll} \textbf{abstractmethod} & \textbf{IBody.remove\_faces}(selection: ansys.geometry.core.designer.face.Face | \\ & collections.abc.Iterable[ansys.geometry.core.designer.face.Face], offset: \\ & ansys.geometry.core.typing.Real) \rightarrow bool \\ \end{tabular}
```

Shell by removing a given set of faces.

### **Parameters**

```
selection
```

[Face | Iterable[Face]] Face or faces to be removed.

offset

[Real] Shell thickness.

### **Returns**

bool

True when successful, False when failed.

```
abstractmethod IBody.plot(merge: bool = True, screenshot: str | None = None, use_trame: bool | None = None, use_service_colors: bool | None = None, **plotting_options: dict | None) \rightarrow None
```

Plot the body.

#### **Parameters**

# merge

[bool, default: True] Whether to merge the body into a single mesh. Performance improved when True. When True (default), the individual faces of the tessellation are merged. When False, the number of triangles are preserved and only the topology is merged.

#### screenshot

[str, default: None] Path for saving a screenshot of the image that is being represented.

#### use trame

[bool, default: None] Whether to enable the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

### use\_service\_colors

[bool, default: None] Whether to use the colors assigned to the body in the service. The default is None, in which case the ansys.geometry.core.USE\_SERVICE\_COLORS global setting is used.

### \*\*plotting\_options

[dict, default: None] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

# **Examples**

Extrude a box centered at the origin to create rectangular body and plot it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> mycomp = design.add_component("my-comp")
>>> body = mycomp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> body.plot()
```

Plot the body and color each face individually:

```
>>> body.plot(multi_colors=True)
```

IBody.intersect(other: Body | collections.abc.Iterable[Body], keep\_other: bool = False)  $\rightarrow$  None Intersect two (or more) bodies.

### **Parameters**

### other

[Body] Body to intersect with.

# keep\_other

[bool, default: False] Whether to retain the intersected body or not.

### Raises

### ValueError

If the bodies do not intersect.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the intersected body is retained.

IBody.**subtract**(other: Body |  $collections.abc.Iterable[Body], <math>keep\_other:$   $bool = False) \rightarrow None$  Subtract two (or more) bodies.

### **Parameters**

#### other

[Body] Body to subtract from the self parameter.

#### keep other

[bool, default: False] Whether to retain the subtracted body or not.

#### Raises

#### ValueError

If the subtraction results in an empty (complete) subtraction.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the subtracted body is retained.

IBody.unite(other: Body | collections.abc.Iterable[Body],  $keep\_other: bool = False$ )  $\rightarrow$  None

Unite two (or more) bodies.

### **Parameters**

#### other

[Body] Body to unite with the self parameter.

# keep\_other

[bool, default: False] Whether to retain the united body or not.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the united body is retained.

### **MasterBody**

```
class ansys.geometry.core.designer.body.MasterBody(id: str, name: str, grpc\_client: ansys.geometry.core.connection.client.GrpcClient, is\_surface: bool = False)
```

Bases: IBody

Represents solids and surfaces organized within the design assembly.

Solids and surfaces synchronize to a design within a supporting Geometry service instance.

### **Parameters**

```
id
```

[str] Server-defined ID for the body.

### name

[str] User-defined label for the body.

# parent\_component

[Component] Parent component to place the new component under within the design assembly.

# grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

### is surface

[bool, default: False] Whether the master body is a surface or an 3D object (with volume). The default is False, in which case the master body is a surface. When True, the master body is a 3D object (with volume).

### **Overview**

# **Abstract methods**

imprint_curves	Imprint all specified geometries onto specified faces of the body.
project_curves	Project all specified geometries onto the body.
<pre>imprint_projected_curves</pre>	Project and imprint specified geometries onto the body.
plot	Plot the body.
intersect	Intersect two (or more) bodies.
subtract	Subtract two (or more) bodies.
unite	Unite two (or more) bodies.

### **Methods**

reset_tessellation_cache	Decorate MasterBody methods that need tessellation cache update.
assign_material	Assign a material against the active design.
<pre>get_assigned_material</pre>	Get the assigned material of the body.
add_midsurface_thickness	Add a mid-surface thickness to a surface body.
add_midsurface_offset	Add a mid-surface offset to a surface body.
translate	Translate the body in a specified direction and distance.
set_name	Set the name of the body.
set_fill_style	Set the fill style of the body.
set_suppressed	Set the body suppression state.
set_color	Set the color of the body.
rotate	Rotate the geometry body around the specified axis by a given angle.
scale	Scale the geometry body by the given value.
map	Map the geometry body to the new specified frame.
mirror	Mirror the geometry body across the specified plane.
<pre>get_collision</pre>	Get the collision state between bodies.
copy	Create a copy of the body under the specified parent.
tessellate	Tessellate the body and return the geometry as triangles.
shell_body	Shell the body to the thickness specified.
remove_faces	Shell by removing a given set of faces.

# **Properties**

id	Get the ID of the body as a string.
name	Get the name of the body.
fill_style	Get the fill style of the body.
is_suppressed	Get the body suppression state.
color	Get the current color of the body.
is_surface	Check if the body is a planar body.
surface_thickness	Get the surface thickness of a surface body.
surface_offset	Get the surface offset type of a surface body.
faces	Get a list of all faces within the body.
edges	Get a list of all edges within the body.
is_alive	Check if the body is still alive and has not been deleted.
volume	Calculate the volume of the body.
material	Get the assigned material of the body.

# **Special methods**

\_\_repr\_\_ Represent the master body as a string.

# Import detail

from ansys.geometry.core.designer.body import MasterBody

# **Property detail**

```
property MasterBody.id: str
Get the ID of the body as a string.

property MasterBody.name: str
Get the name of the body.

property MasterBody.fill_style: str
Get the fill style of the body.

property MasterBody.is_suppressed: bool
Get the body suppression state.

property MasterBody.color: str
Get the current color of the body.

property MasterBody.is_surface: bool
Check if the body is a planar body.
```

property MasterBody.surface\_thickness: pint.Quantity | None

Get the surface thickness of a surface body.

### **Notes**

This method is only for surface-type bodies that have been assigned a surface thickness.

```
{\tt property} \ {\tt MasterBody.surface\_offset:} \ {\tt \textit{MidSurfaceOffsetType}} \ \mid \ {\tt None}
```

Get the surface offset type of a surface body.

### **Notes**

This method is only for surface-type bodies that have been assigned a surface offset.

# property MasterBody.faces: list[ansys.geometry.core.designer.face.Face]

Get a list of all faces within the body.

### **Returns**

list[Face]

### property MasterBody.edges: list[ansys.geometry.core.designer.edge.Edge]

Get a list of all edges within the body.

#### Returns

list[Edge]

# property MasterBody.is\_alive: bool

Check if the body is still alive and has not been deleted.

# property MasterBody.volume: pint.Quantity

Calculate the volume of the body.

### **Notes**

When dealing with a planar surface, a value of 0 is returned as a volume.

# property MasterBody.material: ansys.geometry.core.materials.material.Material

Get the assigned material of the body.

### **Returns**

### **Material**

Material assigned to the body.

# **Method detail**

# MasterBody.reset\_tessellation\_cache()

Decorate MasterBody methods that need tessellation cache update.

### **Parameters**

func

[method] Method to call.

### **Returns**

Any

Output of the method, if any.

 ${\tt MasterBody.} \textbf{assign\_material} (\textit{material:} \ ansys. geometry. core. materials. material. Material) \rightarrow None$ 

Assign a material against the active design.

### **Parameters**

### material

[Material] Source material data.

 ${\tt MasterBody.get\_assigned\_material()} \rightarrow {\it ansys.geometry.core.materials.material.Material}$ 

Get the assigned material of the body.

#### **Returns**

### **Material**

Material assigned to the body.

### $MasterBody.add\_midsurface\_thickness(thickness: pint.Quantity) \rightarrow None$

Add a mid-surface thickness to a surface body.

#### **Parameters**

### thickness

[Quantity] Thickness to assign.

### **Notes**

Only surface bodies are eligible for mid-surface thickness assignment.

 $\texttt{MasterBody.add\_midsurface\_offset}(\textit{offset}: MidSurfaceOffsetType}) \rightarrow \texttt{None}$ 

Add a mid-surface offset to a surface body.

### **Parameters**

### offset\_type

[MidSurfaceOffsetType] Surface offset to assign.

### **Notes**

Only surface bodies are eligible for mid-surface offset assignment.

abstractmethod MasterBody.imprint\_curves(faces: list[ansys.geometry.core.designer.face.Face], sketch:

ansys.geometry.core.sketch.sketch.Sketch)  $\rightarrow$  tuple[list[ansys.geometry.core.designer.edge.Edge], list[ansys.geometry.core.designer.face.Face]]

Imprint all specified geometries onto specified faces of the body.

### **Parameters**

### faces: list[Face]

list of faces to imprint the curves of the sketch onto.

sketch: Sketch

All curves to imprint on the faces.

### Returns

```
tuple[list[Edge], list[Face]]
```

All impacted edges and faces from the imprint operation.

**abstractmethod** MasterBody.project\_curves(direction: ansys.geometry.core.math.vector.UnitVector3D,

sketch: ansys.geometry.core.sketch.sketch.Sketch, closest\_face: bool, only\_one\_curve: bool = False)  $\rightarrow$ 

list[ansys.geometry.core.designer.face.Face]

Project all specified geometries onto the body.

### **Parameters**

direction: UnitVector3D

Direction of the projection.

sketch: Sketch

All curves to project on the body.

closest\_face: bool

Whether to target the closest face with the projection.

#### only one curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### Returns

### list[Face]

All faces from the project curves operation.

### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

# abstractmethod MasterBody.imprint\_projected\_curves(direction:

ansys.geometry.core.math.vector.UnitVector3D, *sketch:* ansys.geometry.core.sketch.sketch.Sketch, *closest\_face: bool, only\_one\_curve: bool = False)* → list[ansys.geometry.core.designer.face.Face]

Project and imprint specified geometries onto the body.

This method combines the project\_curves() and imprint\_curves() method into one method. It has higher performance than calling them back-to-back when dealing with many curves. Because it is a specialized function, this method only returns the faces (and not the edges) from the imprint operation.

#### **Parameters**

#### direction: UnitVector3D

Direction of the projection.

# sketch: Sketch

All curves to project on the body.

### closest face: bool

Whether to target the closest face with the projection.

### only\_one\_curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### Returns

# list[Face]

All imprinted faces from the operation.

### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

 $\label{lem:masterBody.translate} \begin{tabular}{ll} MasterBody.translate({\it direction:} ansys.geometry.core.math.vector.UnitVector3D, {\it distance:} pint.Quantity | ansys.geometry.core.misc.measurements.Distance | {\it ansys.geometry.core.typing.Real}) $\rightarrow $$ None $$$ 

Translate the body in a specified direction and distance.

### **Parameters**

### direction: UnitVector3D

Direction of the translation.

# distance: ~pint.Quantity | Distance | Real

Distance (magnitude) of the translation.

#### Returns

None

MasterBody.set\_name(name: str)  $\rightarrow$  None

Set the name of the body.

 $MasterBody.set_fill_style(fill_style: FillStyle) \rightarrow None$ 

Set the fill style of the body.

MasterBody.set\_suppressed(suppressed: bool)  $\rightarrow$  None

Set the body suppression state.

MasterBody.set\_color( $color: str \mid tuple[float, float]$ )  $\rightarrow$  None

Set the color of the body.

MasterBody.rotate(axis\_origin: ansys.geometry.core.math.point.Point3D, axis\_direction:

ansys.geometry.core.math.vector.UnitVector3D, angle: pint.Quantity |

ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real)  $\rightarrow$  None

Rotate the geometry body around the specified axis by a given angle.

#### **Parameters**

axis\_origin: Point3D

Origin of the rotational axis.

axis\_direction: UnitVector3D

The axis of rotation.

angle: ~pint.Quantity | Angle | Real

Angle (magnitude) of the rotation.

### Returns

None

 $\texttt{MasterBody.scale}(\textit{value: ansys.geometry.core.typing.Real}) \rightarrow \texttt{None}$ 

Scale the geometry body by the given value.

### **Parameters**

value: Real

Value to scale the body by.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

 ${\tt MasterBody.map}(frame: ansys.geometry.core.math.frame.Frame) o {\tt None}$ 

Map the geometry body to the new specified frame.

### **Parameters**

frame: Frame

Structure defining the orientation of the body.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

MasterBody.mirror(plane: ansys.geometry.core.math.plane.Plane) o None

Mirror the geometry body across the specified plane.

### **Parameters**

### plane: Plane

Represents the mirror.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

MasterBody.get\_collision(body: Body)  $\rightarrow CollisionType$ 

Get the collision state between bodies.

#### **Parameters**

# body: Body

Object that the collision state is checked with.

### Returns

### CollisionType

Enum that defines the collision state between bodies.

MasterBody.copy(parent: ansys.geometry.core.designer.component.Component, name: str = None)  $\rightarrow Body$ Create a copy of the body under the specified parent.

### **Parameters**

### parent: Component

Parent component to place the new body under within the design assembly.

#### name: str

Name to give the new body.

# Returns

### **Body**

Copy of the body.

MasterBody.tessellate(merge: bool = False, transform: ansys.geometry.core.math.matrix.Matrix44 =  $IDENTITY\_MATRIX44$ )  $\rightarrow$  pyvista.PolyData | pyvista.MultiBlock

Tessellate the body and return the geometry as triangles.

### **Parameters**

#### merge

[bool, default: False] Whether to merge the body into a single mesh. When False (default), the number of triangles are preserved and only the topology is merged. When True, the individual faces of the tessellation are merged.

### Returns

### PolyData, MultiBlock

Merged pyvista.PolyData if merge=True or a composite dataset.

### **Examples**

Extrude a box centered at the origin to create a rectangular body and tessellate it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> my_comp = design.add_component("my-comp")
>>> body = my_comp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> blocks = body.tessellate()
>>> blocks
>>> MultiBlock(0x7F94EC757460)
    N Blocks: 6
    X Bounds: 0.000, 4.000
    Y Bounds: -1.000, 0.000
     Z Bounds: -0.500, 4.500
```

Merge the body:

 $\texttt{MasterBody.shell\_body}(\textit{offset: ansys.geometry.core.typing.Real}) \rightarrow \texttt{bool}$ 

Shell the body to the thickness specified.

### **Parameters**

### offset

[Real] Shell thickness.

### Returns

bool

True when successful, False when failed.

```
\label{lem:masterBody.remove_faces} $$ (selection: ansys.geometry.core.designer.face.Face | $$ collections.abc.Iterable[ansys.geometry.core.designer.face.Face], offset: $$ ansys.geometry.core.typing.Real) $$ \to bool$
```

Shell by removing a given set of faces.

### **Parameters**

### selection

[Face | Iterable[Face]] Face or faces to be removed.

### offset

[Real] Shell thickness.

#### Returns

bool

True when successful, False when failed.

```
abstractmethod MasterBody.plot(merge: bool = True, screenshot: str \mid None = None, use_trame: bool | None = None, use_service_colors: bool | None = None, **plotting_options: dict | None) \rightarrow None
```

Plot the body.

### **Parameters**

#### merge

[bool, default: True] Whether to merge the body into a single mesh. Performance improved when True. When True (default), the individual faces of the tessellation are merged. When False, the number of triangles are preserved and only the topology is merged.

### screenshot

[str, default: None] Path for saving a screenshot of the image that is being represented.

#### use trame

[bool, default: None] Whether to enable the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

### use\_service\_colors

[bool, default: None] Whether to use the colors assigned to the body in the service. The default is None, in which case the ansys.geometry.core.USE\_SERVICE\_COLORS global setting is used.

### \*\*plotting\_options

[dict, default: None] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

### **Examples**

Extrude a box centered at the origin to create rectangular body and plot it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> mycomp = design.add_component("my-comp")
>>> body = mycomp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> body.plot()
```

Plot the body and color each face individually:

```
>>> body.plot(multi_colors=True)
```

abstractmethod MasterBody.intersect(other: Body | collections.abc.Iterable[Body], keep\_other: bool = False)  $\rightarrow$  None

Intersect two (or more) bodies.

#### **Parameters**

other

[Body] Body to intersect with.

### keep\_other

[bool, default: False] Whether to retain the intersected body or not.

### Raises

### ValueError

If the bodies do not intersect.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the intersected body is retained.

**abstractmethod** MasterBody.**subtract**( $other: Body \mid collections.abc.Iterable[Body], keep_other: bool = False) <math>\rightarrow$  None

Subtract two (or more) bodies.

### **Parameters**

other

[Body] Body to subtract from the self parameter.

### keep\_other

[bool, default: False] Whether to retain the subtracted body or not.

### **Raises**

### ValueError

If the subtraction results in an empty (complete) subtraction.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the subtracted body is retained.

Unite two (or more) bodies.

### **Parameters**

other

[Body] Body to unite with the self parameter.

#### keep\_other

[bool, default: False] Whether to retain the united body or not.

# Notes

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the united body is retained.

```
MasterBody.__repr__() \rightarrow str
```

Represent the master body as a string.

### **Body**

class ansys.geometry.core.designer.body.Body(id, name, parent\_component:

ansys.geometry.core.designer.component.Component,

template: MasterBody)

Bases: IBody

Represents solids and surfaces organized within the design assembly.

Solids and surfaces synchronize to a design within a supporting Geometry service instance.

### **Parameters**

id

[str] Server-defined ID for the body.

#### name

[str] User-defined label for the body.

# parent\_component

[Component] Parent component to place the new component under within the design assembly.

# template

[MasterBody] Master body that this body is an occurrence of.

# **Overview**

# Methods

reset_tessellation_cache	Decorate Body methods that require a tessellation cache update.
assign_material	Assign a material against the active design.
<pre>get_assigned_material</pre>	Get the assigned material of the body.
add_midsurface_thickness	Add a mid-surface thickness to a surface body.
add_midsurface_offset	Add a mid-surface offset to a surface body.
imprint_curves	Imprint all specified geometries onto specified faces of the body.
project_curves	Project all specified geometries onto the body.
<pre>imprint_projected_curves</pre>	Project and imprint specified geometries onto the body.
set_name	Set the name of the body.
set_fill_style	Set the fill style of the body.
set_suppressed	Set the body suppression state.
set_color	Set the color of the body.
translate	Translate the body in a specified direction and distance.
rotate	Rotate the geometry body around the specified axis by a given angle.
scale	Scale the geometry body by the given value.
map	Map the geometry body to the new specified frame.
mirror	Mirror the geometry body across the specified plane.
get_collision	Get the collision state between bodies.
copy	Create a copy of the body under the specified parent.
tessellate	Tessellate the body and return the geometry as triangles.
shell_body	Shell the body to the thickness specified.
remove_faces	Shell by removing a given set of faces.
plot	Plot the body.
intersect	Intersect two (or more) bodies.
subtract	Subtract two (or more) bodies.
unite	Unite two (or more) bodies.

# **Properties**

id	Get the ID of the body as a string.
name	Get the name of the body.
fill_style	Get the fill style of the body.
is_suppressed	Get the body suppression state.
color	Get the color of the body.
parent_component	
faces	Get a list of all faces within the body.
edges	Get a list of all edges within the body.
is_alive	Check if the body is still alive and has not been deleted.
is_surface	Check if the body is a planar body.
surface_thickness	Get the surface thickness of a surface body.
surface_offset	Get the surface offset type of a surface body.
volume	Calculate the volume of the body.
material	Get the assigned material of the body.

# Special methods

repr Represent the Body as a string
-------------------------------------

# Import detail

```
from ansys.geometry.core.designer.body import Body
Property detail
property Body.id: str
     Get the ID of the body as a string.
property Body.name: str
     Get the name of the body.
property Body.fill_style: str
     Get the fill style of the body.
property Body.is_suppressed: bool
     Get the body suppression state.
property Body.color: str
     Get the color of the body.
property Body.parent_component: ansys.geometry.core.designer.component.Component
property Body.faces: list[ansys.geometry.core.designer.face.Face]
     Get a list of all faces within the body.
          Returns
              list[Face]
property Body.edges: list[ansys.geometry.core.designer.edge.Edge]
     Get a list of all edges within the body.
          Returns
              list[Edge]
property Body.is_alive: bool
     Check if the body is still alive and has not been deleted.
property Body.is_surface: bool
     Check if the body is a planar body.
property Body.surface_thickness: pint.Quantity | None
     Get the surface thickness of a surface body.
     Notes
     This method is only for surface-type bodies that have been assigned a surface thickness.
property Body.surface_offset: MidSurfaceOffsetType | None
     Get the surface offset type of a surface body.
     Notes
     This method is only for surface-type bodies that have been assigned a surface offset.
```

**property** Body.**volume: pint.Quantity**Calculate the volume of the body.

### **Notes**

When dealing with a planar surface, a value of 0 is returned as a volume.

# property Body.material: ansys.geometry.core.materials.material.Material

Get the assigned material of the body.

### **Returns**

### **Material**

Material assigned to the body.

### **Method detail**

# Body.reset\_tessellation\_cache()

Decorate Body methods that require a tessellation cache update.

### **Parameters**

func

[method] Method to call.

#### Returns

Any

Output of the method, if any.

 $Body. \textbf{assign\_material}. \textit{(material: ansys.geometry.core.materials.material.Material)} \rightarrow None$ 

Assign a material against the active design.

### **Parameters**

### material

[Material] Source material data.

 ${\tt Body.get\_assigned\_material()} \rightarrow {\it ansys.geometry.core.materials.material.Material}$ 

Get the assigned material of the body.

### Returns

### **Material**

Material assigned to the body.

 ${\tt Body.add\_midsurface\_thickness}(\textit{thickness: pint.Quantity}) \rightarrow {\tt None}$ 

Add a mid-surface thickness to a surface body.

#### **Parameters**

### thickness

[Quantity] Thickness to assign.

### **Notes**

Only surface bodies are eligible for mid-surface thickness assignment.

Body.add\_midsurface\_offset(offset: MidSurfaceOffsetType)  $\rightarrow$  None

Add a mid-surface offset to a surface body.

# **Parameters**

### offset type

[MidSurfaceOffsetType] Surface offset to assign.

### **Notes**

Only surface bodies are eligible for mid-surface offset assignment.

Body.**imprint\_curves**( $faces: list[ansys.geometry.core.designer.face.Face], <math>sketch: ansys.geometry.core.sketch.Sketch) \rightarrow$ 

tuple[list[ansys.geometry.core.designer.edge.Edge],

 $list [{\it ansys.geometry.core.designer.face.Face}]]$ 

Imprint all specified geometries onto specified faces of the body.

#### **Parameters**

faces: list[Face]

list of faces to imprint the curves of the sketch onto.

sketch: Sketch

All curves to imprint on the faces.

### Returns

```
tuple[list[Edge], list[Face]]
```

All impacted edges and faces from the imprint operation.

Body.project\_curves(direction: ansys.geometry.core.math.vector.UnitVector3D, sketch:

ansys.geometry.core.sketch.sketch.sketch,  $closest\_face: bool, only\_one\_curve: bool = False) \rightarrow list[ansys.geometry.core.designer.face.Face]$ 

Project all specified geometries onto the body.

### **Parameters**

direction: UnitVector3D

Direction of the projection.

sketch: Sketch

All curves to project on the body.

closest face: bool

Whether to target the closest face with the projection.

only\_one\_curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### **Returns**

#### list[Face]

All faces from the project curves operation.

### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

Body.imprint\_projected\_curves(direction: ansys.geometry.core.math.vector.UnitVector3D, sketch:

ansys.geometry.core.sketch.sketch.Sketch, closest\_face: bool,

 $only\_one\_curve: bool = False) \rightarrow$ 

 $list [{\it ansys.geometry.core.designer.face.Face}]$ 

Project and imprint specified geometries onto the body.

This method combines the project\_curves() and imprint\_curves() method into one method. It has higher performance than calling them back-to-back when dealing with many curves. Because it is a specialized function, this method only returns the faces (and not the edges) from the imprint operation.

#### **Parameters**

### direction: UnitVector3D

Direction of the projection.

sketch: Sketch

All curves to project on the body.

closest face: bool

Whether to target the closest face with the projection.

only\_one\_curve: bool, default: False

Whether to project only one curve of the entire sketch. When True, only one curve is projected.

### **Returns**

### list[Face]

All imprinted faces from the operation.

#### **Notes**

The only\_one\_curve parameter allows you to optimize the server call because projecting curves is an expensive operation. This reduces the workload on the server side.

Body.set\_name(name: str)  $\rightarrow$  None

Set the name of the body.

Body.set\_fill\_style( $fill_style$ : FillStyle)  $\rightarrow$  None

Set the fill style of the body.

Body.set\_suppressed(suppressed: bool)  $\rightarrow$  None

Set the body suppression state.

Body.set\_color(color: str | tuple[float, float, float])  $\rightarrow$  None

Set the color of the body.

### **Parameters**

### color

[str | tuple[float, float, float]] Color to set the body to. This can be a string representing a color name or a tuple of RGB values in the range [0, 1] (RGBA) or [0, 255] (pure RGB).

Body.translate(direction: ansys.geometry.core.math.vector.UnitVector3D, distance: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real)  $\rightarrow$  None

Translate the body in a specified direction and distance.

### **Parameters**

direction: UnitVector3D

Direction of the translation.

**distance:** ~pint.Quantity | Distance | Real Distance (magnitude) of the translation.

Returns

None

```
Body.rotate(axis\_origin: ansys.geometry.core.math.point.Point3D, axis\_direction: ansys.geometry.core.math.vector.UnitVector3D, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real) \rightarrow None
```

Rotate the geometry body around the specified axis by a given angle.

#### **Parameters**

axis\_origin: Point3D

Origin of the rotational axis.

axis\_direction: UnitVector3D

The axis of rotation.

angle: ~pint.Quantity | Angle | Real Angle (magnitude) of the rotation.

#### Returns

#### None

Body.scale(value: ansys.geometry.core.typing.Real)  $\rightarrow$  None

Scale the geometry body by the given value.

### **Parameters**

value: Real

Value to scale the body by.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

Body.map(frame: ansys.geometry.core.math.frame.Frame)  $\rightarrow$  None

Map the geometry body to the new specified frame.

### **Parameters**

### frame: Frame

Structure defining the orientation of the body.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

Body.mirror(plane: ansys.geometry.core.math.plane.Plane)  $\rightarrow$  None

Mirror the geometry body across the specified plane.

### **Parameters**

# plane: Plane

Represents the mirror.

### **Notes**

The calling object is directly modified when this method is called. Thus, it is important to make copies if needed.

 ${\tt Body.get\_collision}(body: \ {\tt Body}) \rightarrow {\it CollisionType}$ 

Get the collision state between bodies.

### **Parameters**

### body: Body

Object that the collision state is checked with.

### Returns

### CollisionType

Enum that defines the collision state between bodies.

Body.copy(parent: ansys.geometry.core.designer.component.Component, name: str = None)  $\rightarrow Body$ Create a copy of the body under the specified parent.

### **Parameters**

# parent: Component

Parent component to place the new body under within the design assembly.

#### name: str

Name to give the new body.

### **Returns**

#### **Body**

Copy of the body.

Body.tessellate(merge: bool = False)  $\rightarrow$  pyvista.PolyData|pyvista.MultiBlock

Tessellate the body and return the geometry as triangles.

#### **Parameters**

### merge

[bool, default: False] Whether to merge the body into a single mesh. When False (default), the number of triangles are preserved and only the topology is merged. When True, the individual faces of the tessellation are merged.

#### Returns

# PolyData, MultiBlock

Merged pyvista.PolyData if merge=True or a composite dataset.

# **Examples**

Extrude a box centered at the origin to create a rectangular body and tessellate it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> my_comp = design.add_component("my-comp")
>>> body = my_comp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> blocks = body.tessellate()
>>> blocks
>>> MultiBlock(0x7F94EC757460)
    N Blocks: 6
    X Bounds: 0.000, 4.000
    Y Bounds: -1.000, 0.000
     Z Bounds: -0.500, 4.500
```

# Merge the body:

Body.**shell\_body**( $offset: ansys.geometry.core.typing.Real) <math>\rightarrow$  bool

Shell the body to the thickness specified.

# **Parameters**

### offset

[Real] Shell thickness.

#### Returns

bool

True when successful, False when failed.

Body.**remove\_faces**(selection: ansys.geometry.core.designer.face.Face | collections.abc.Iterable[ansys.geometry.core.designer.face.Face], offset: ansys.geometry.core.typing.Real) op bool

Shell by removing a given set of faces.

### **Parameters**

#### selection

[Face | Iterable[Face]] Face or faces to be removed.

### offset

[Rea1] Shell thickness.

### Returns

bool

True when successful, False when failed.

```
Body.plot(merge: bool = True, screenshot: str | None = None, use_trame: bool | None = None, use_service_colors: bool | None = None, **plotting_options: dict | None) \rightarrow None
```

Plot the body.

# **Parameters**

### merge

[bool, default: True] Whether to merge the body into a single mesh. Performance improved when True. When True (default), the individual faces of the tessellation are merged. When False, the number of triangles are preserved and only the topology is merged.

#### screenshot

[str, default: None] Path for saving a screenshot of the image that is being represented.

### use trame

[bool, default: None] Whether to enable the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

#### use service colors

[bool, default: None] Whether to use the colors assigned to the body in the service. The default is None, in which case the ansys.geometry.core.USE\_SERVICE\_COLORS global setting is used.

# \*\*plotting\_options

[dict, default: None] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

# **Examples**

Extrude a box centered at the origin to create rectangular body and plot it:

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 0, 1])
>>> sketch = Sketch(plane)
>>> box = sketch.box(Point2D([2, 0]), 4, 4)
>>> design = modeler.create_design("my-design")
>>> mycomp = design.add_component("my-comp")
>>> body = mycomp.extrude_sketch("my-sketch", sketch, 1 * u.m)
>>> body.plot()
```

Plot the body and color each face individually:

```
>>> body.plot(multi_colors=True)
```

Body.intersect(other: Body | collections.abc.Iterable[Body], keep\_other: bool = False)  $\rightarrow$  None Intersect two (or more) bodies.

#### **Parameters**

#### other

[Body] Body to intersect with.

### keep\_other

[bool, default: False] Whether to retain the intersected body or not.

### Raises

#### ValueError

If the bodies do not intersect.

#### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the intersected body is retained.

Body. **subtract**(*other*: Body | *collections.abc.lterable*[Body],  $keep\_other: bool = False$ )  $\rightarrow$  None Subtract two (or more) bodies.

### **Parameters**

#### other

[Body] Body to subtract from the self parameter.

### keep\_other

[bool, default: False] Whether to retain the subtracted body or not.

#### Raises

### ValueError

If the subtraction results in an empty (complete) subtraction.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the subtracted body is retained.

Body.unite(other: Body | collections.abc.Iterable[Body],  $keep\_other: bool = False$ )  $\rightarrow$  None

Unite two (or more) bodies.

#### **Parameters**

### other

[Body] Body to unite with the self parameter.

### keep\_other

[bool, default: False] Whether to retain the united body or not.

### **Notes**

The self parameter is directly modified with the result, and the other parameter is consumed. Thus, it is important to make copies if needed. If the keep\_other parameter is set to True, the united body is retained.

```
Body.__repr__() \rightarrow str
```

Represent the Body as a string.

### MidSurfaceOffsetType

class ansys.geometry.core.designer.body.MidSurfaceOffsetType(\*args, \*\*kwds)

Bases: enum. Enum

Provides values for mid-surface offsets supported.

# **Overview**

### **Attributes**

MIDDLE TOP BOTTOM VARIABLE CUSTOM

# Import detail

from ansys.geometry.core.designer.body import MidSurfaceOffsetType

# **Attribute detail**

Provides values for collision types between bodies.

### Overview

### **Attributes**

NONE
TOUCH
INTERSECT
CONTAINED
CONTAINEDTOUCH

# Import detail

```
from ansys.geometry.core.designer.body import CollisionType
```

### Attribute detail

Provides values for fill styles supported.

# **Overview**

### **Attributes**

DEFAULT OPAQUE TRANSPARENT

# Import detail

from ansys.geometry.core.designer.body import FillStyle

# **Attribute detail**

FillStyle.**DEFAULT** = **0** 

FillStyle.OPAQUE = 1

FillStyle.TRANSPARENT = 2

# **Description**

Provides for managing a body.

# **Module detail**

body.\_\_TEMPORARY\_BOOL\_OPS\_FIX\_\_ = (99, 0, 0)

The component.py module

# **Summary**

# Classes

Component Provides for creating and managing a component.

### **Enums**

SharedTopologyType	Shared topologies available.
ExtrusionDirection	Enum for extrusion direction definition.

# Component

class ansys.geometry.core.designer.component.Component(name: str, parent\_component: Component)

None, grpc\_client: an-

sys.geometry.core.connection.client.GrpcClient,

template: Component | None = None, instance\_name: str | None = None, preexisting id: str | None = None,

master\_component: an-

sys.geometry.core.designer.part.MasterComponent | None = None, read existing comp: bool =

False)

Provides for creating and managing a component.

This class synchronizes to a design within a supporting Geometry service instance.

### **Parameters**

#### name

[str] User-defined label for the new component.

### parent\_component

[Component or None] Parent component to place the new component under within the design assembly. The default is None only when dealing with a Design object.

# grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

### template

[Component, default: None] Template to create this component from. This creates an instance component that shares a master with the template component.

# instance\_name: str, default: None

User defined optional name for the component instance.

### preexisting\_id

[str, default: None] ID of a component pre-existing on the server side to use to create the component on the client-side data model. If an ID is specified, a new component is not created on the server.

### master\_component

[MasterComponent, default: None] Master component to use to create a nested component instance instead of creating a new conponent.

### read\_existing\_comp

[bool, default: False] Whether an existing component on the service should be read. This parameter is only valid when connecting to an existing service session. Otherwise, avoid using this optional parameter.

### Overview

# **Methods**

get_all_bodies	Get all bodies in the component hierarchy.
<pre>get_world_transform</pre>	Get the full transformation matrix of the component in world space.
modify_placement	Apply a translation and/or rotation to the placement matrix.
reset_placement	Reset a component's placement matrix to an identity matrix.
add_component	Add a new component under this component within the design assembly.
set_shared_topology	Set the shared topology to apply to the component.

continues on next page

Table 2 – continued from previous page

extrude_sketch	Create a solid body by extruding the sketch profile a distance.
sweep_sketch	Create a body by sweeping a planar profile along a path.
sweep_chain	Create a body by sweeping a chain of curves along a path.
revolve_sketch	Create a solid body by revolving a sketch profile around an axis.
extrude_face	Extrude the face profile by a given distance to create a solid body.
create_sphere	Create a sphere body defined by the center point and the radius.
<pre>create_body_from_loft_profile</pre>	Create a lofted body from a collection of trimmed curves.
create_surface	Create a surface body with a sketch profile.
create_surface_from_face	Create a surface body based on a face.
create_body_from_surface	Create a surface body from a trimmed surface.
<pre>create_surface_from_trimmed_curves</pre>	Create a surface body from a list of trimmed curves all lying on the same plane.
create_coordinate_system	Create a coordinate system.
translate_bodies	Translate the bodies in a specified direction by a distance.
create_beams	Create beams under the component.
create_beam	Create a beam under the component.
delete_component	Delete a component (itself or its children).
delete_body	Delete a body belonging to this component (or its children).
add_design_point	Create a single design point.
add_design_points	Create a list of design points.
delete_beam	Delete an existing beam belonging to this component's scope.
search_component	Search nested components recursively for a component.
search_body	Search bodies in the component's scope.
search_beam	Search beams in the component's scope.
tessellate	Tessellate the component.
plot	Plot the component.
tree_print	Print the component in tree format.

# **Properties**

id	ID of the component.
name	Name of the component.
instance_name	Name of the component instance.
components	List of Component objects inside of the component.
bodies	List of Body objects inside of the component.
beams	List of Beam objects inside of the component.
design_points	List of DesignPoint objects inside of the component.
coordinate_systems	List of CoordinateSystem objects inside of the component.
parent_component	Parent of the component.
is_alive	Whether the component is still alive on the server side.
shared_topology	Shared topology type of the component (if any).

# **Special methods**

\_\_repr\_\_ Represent the Component as a string.

# Import detail

```
from ansys.geometry.core.designer.component import Component
Property detail
property Component.id: str
     ID of the component.
property Component.name: str
     Name of the component.
property Component.instance_name: str
     Name of the component instance.
property Component.components: list[Component]
     List of Component objects inside of the component.
property Component.bodies: list[ansys.geometry.core.designer.body.Body]
     List of Body objects inside of the component.
property Component.beams: list[ansys.geometry.core.designer.beam.Beam]
     List of Beam objects inside of the component.
property Component.design_points:
list[ansys.geometry.core.designer.designpoint.DesignPoint]
     List of DesignPoint objects inside of the component.
property Component.coordinate_systems:
list[ansys.geometry.core.designer.coordinate_system.CoordinateSystem]
     List of CoordinateSystem objects inside of the component.
property Component.parent_component: Component
     Parent of the component.
property Component.is_alive: bool
     Whether the component is still alive on the server side.
property Component.shared_topology: SharedTopologyType | None
     Shared topology type of the component (if any).
     Notes
     If no shared topology has been set, None is returned.
Method detail
Component.get_all_bodies() \rightarrow list[ansys.geometry.core.designer.body.Body]
     Get all bodies in the component hierarchy.
```

# Returns

list[Body]

List of all bodies in the component hierarchy.

### Component.get\_world\_transform() $\rightarrow$ ansys.geometry.core.math.matrix.Matrix44

Get the full transformation matrix of the component in world space.

#### Returns

#### Matrix44

4x4 transformation matrix of the component in world space.

```
Component.modify_placement(translation: ansys.geometry.core.math.vector.Vector3D | None = None, rotation_origin: ansys.geometry.core.math.point.Point3D | None = None, rotation_direction: ansys.geometry.core.math.vector.UnitVector3D | None = None, rotation_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0)
```

Apply a translation and/or rotation to the placement matrix.

#### **Parameters**

### translation

[Vector3D, default: None] Vector that defines the desired translation to the component.

### rotation origin

[Point3D, default: None] Origin that defines the axis to rotate the component about.

#### rotation\_direction

[UnitVector3D, default: None] Direction of the axis to rotate the component about.

### rotation\_angle

[Quantity | Angle | Real, default: 0] Angle to rotate the component around the axis.

# **Notes**

To reset a component's placement to an identity matrix, see <code>reset\_placement()</code> or call <code>modify\_placement()</code> with no arguments.

### Component.reset\_placement()

Reset a component's placement matrix to an identity matrix.

See modify\_placement().

Component . add\_component (name: str, template: Component | None = None,  $instance\_name$ : str = None)  $\rightarrow$  Component

Add a new component under this component within the design assembly.

### **Parameters**

#### name

[str] User-defined label for the new component.

#### template

[Component, default: None] Template to create this component from. This creates an instance component that shares a master with the template component.

# Returns

### Component

New component with no children in the design assembly.

# Component.set\_shared\_topology( $share\_type$ : SharedTopologyType) $\rightarrow$ None

Set the shared topology to apply to the component.

#### **Parameters**

#### share type

[SharedTopologyType] Shared topology type to assign to the component.

Component.extrude\_sketch(name: str, sketch: ansys.geometry.core.sketch.sketch.Sketch, distance:  $pint.Quantity \mid$  ansys.geometry.core.misc.measurements.Distance  $\mid$  ansys.geometry.core.typing.Real, direction: ExtrusionDirection  $\mid str = ExtrusionDirection.POSITIVE$ , cut: bool = False)  $\rightarrow$  ansys.geometry.core.designer.body.Body  $\mid$  None

Create a solid body by extruding the sketch profile a distance.

#### **Parameters**

#### name

[str] User-defined label for the new solid body.

#### sketch

[Sketch] Two-dimensional sketch source for the extrusion.

### distance

[Quantity | Distance | Real] Distance to extrude the solid body.

#### direction

[ExtrusionDirection | str, default: "+"] Direction for extruding the solid body. The default is to extrude in the positive normal direction of the sketch. Options are "+" and "-" as a string, or the enum values.

#### cut

[bool, default: False] Whether to cut the extrusion from the existing component. By default, the extrusion is added to the existing component.

#### Returns

### **Body**

Extruded body from the given sketch.

# None

If the cut parameter is True, the function returns None.

# **Notes**

The newly created body is placed under this component within the design assembly.

```
Component.sweep_sketch(name: str, sketch: ansys.geometry.core.sketch.sketch.Sketch, path: list[ansys.geometry.core.shapes.curves.trimmed_curve.TrimmedCurve]) \rightarrow ansys.geometry.core.designer.body.Body
```

Create a body by sweeping a planar profile along a path.

#### **Parameters**

#### name

[str] User-defined label for the new solid body.

# sketch

[Sketch] Two-dimensional sketch source for the extrusion.

### path

[list[TrimmedCurve]] The path to sweep the profile along.

#### Returns

# **Body**

Created body from the given sketch.

#### **Notes**

The newly created body is placed under this component within the design assembly.

```
\label{list_ansys} \begin{subarray}{l} Component.sweep\_chain(name: str, path: \\ list[ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve], chain: \\ list[ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve]) \rightarrow \\ ansys.geometry.core.designer.body.Body \end{subarray}
```

Create a body by sweeping a chain of curves along a path.

```
Parameters
```

```
name
    [str] User-defined label for the new solid body.
path
    [list[TrimmedCurve]] The path to sweep the chain along.
chain
    [list[TrimmedCurve]] A chain of trimmed curves.
```

#### **Returns**

**Body** 

Created body from the given sketch.

#### **Notes**

The newly created body is placed under this component within the design assembly.

```
Component.revolve_sketch(name: str, sketch: ansys.geometry.core.sketch.sketch.sketch.sketch, axis: ansys.geometry.core.math.vector.Vector3D, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, rotation_origin: ansys.geometry.core.math.point.Point3D) \rightarrow ansys.geometry.core.designer.body.Body
```

Create a solid body by revolving a sketch profile around an axis.

# **Parameters**

```
name
```

[str] User-defined label for the new solid body.

#### sketch

[Sketch] Two-dimensional sketch source for the revolve.

# axis

[Vector3D] Axis of rotation for the revolve.

### angle

[Quantity | Angle | Real] Angle to revolve the solid body around the axis. The angle can be positive or negative.

# rotation\_origin

[Point3D] Origin of the axis of rotation.

# Returns

**Body** 

Revolved body from the given sketch.

Component.extrude\_face(name: str, face: ansys.geometry.core.designer.face.Face, distance: pint.Quantity | ansys.geometry.core.misc.measurements.Distance, direction: ExtrusionDirection | str = ExtrusionDirection.POSITIVE)  $\rightarrow$  ansys.geometry.core.designer.body.Body

Extrude the face profile by a given distance to create a solid body.

There are no modifications against the body containing the source face.

#### **Parameters**

# name

[str] User-defined label for the new solid body.

#### face

[Face] Target face to use as the source for the new surface.

#### distance

[Quantity | Distance | Real] Distance to extrude the solid body.

### direction

[ExtrusionDirection | str, default: "+"] Direction for extruding the solid body's face. The default is to extrude in the positive normal direction of the face. Options are "+" and "-" as a string, or the enum values.

#### Returns

# Body

Extruded solid body.

#### **Notes**

The source face can be anywhere within the design component hierarchy. Therefore, there is no validation requiring that the face is placed under the target component where the body is to be created.

```
Component.create_sphere(name: str, center: ansys.geometry.core.math.point.Point3D, radius: ansys.geometry.core.misc.measurements.Distance) \rightarrow ansys.geometry.core.designer.body.Body
```

Create a sphere body defined by the center point and the radius.

#### **Parameters**

```
name
```

[str] Body name.

#### center

[Point3D] Center point of the sphere.

### radius

[Distance] Radius of the sphere.

#### Returns

#### **Body**

Sphere body object.

# Component.create\_body\_from\_loft\_profile(name: str, profiles:

 $list[list[ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve]],$   $periodic: bool = False, ruled: bool = False) \rightarrow ansys.geometry.core.designer.body.Body$ 

Create a lofted body from a collection of trimmed curves.

#### **Parameters**

#### name

[str] Name of the lofted body.

# profiles

[list[list[TrimmedCurve]]] Collection of lists of trimmed curves (profiles) defining the lofted body's shape.

### periodic

[bool, default: False] Whether the lofted body should have periodic continuity.

#### ruled

[bool] Whether the lofted body should be ruled.

#### Returns

### **Body**

Created lofted body object.

#### **Notes**

Surfaces produced have a U parameter in the direction of the profile curves, and a V parameter in the direction of lofting. Profiles can have different numbers of segments. A minimum twist solution is produced. Profiles should be all closed or all open. Closed profiles cannot contain inner loops. If closed profiles are supplied, a closed (solid) body is produced, if possible. Otherwise, an open (sheet) body is produced. The periodic argument applies when the profiles are closed. It is ignored if the profiles are open.

If periodic=True, at least three profiles must be supplied. The loft continues from the last profile back to the first profile to produce surfaces that are periodic in V.

If periodic=False, at least two profiles must be supplied. If the first and last profiles are planar, end capping faces are created. Otherwise, an open (sheet) body is produced. If ruled=True, separate ruled surfaces are produced between each pair of profiles. If periodic=True, the loft continues from the last profile back to the first profile, but the surfaces are not periodic.

```
Component.create_surface(name: str, sketch: ansys.geometry.core.sketch.sketch.sketch) \rightarrow ansys.geometry.core.designer.body.Body
```

Create a surface body with a sketch profile.

The newly created body is placed under this component within the design assembly.

#### **Parameters**

#### name

[str] User-defined label for the new surface body.

# sketch

[Sketch] Two-dimensional sketch source for the surface definition.

# **Returns**

#### Body

Body (as a planar surface) from the given sketch.

 $\label{local_component} \textbf{Create\_surface\_from\_face}(\textit{name: str, face: ansys.geometry.core.designer.face.Face}) \rightarrow \textit{ansys.geometry.core.designer.body.Body}$ 

Create a surface body based on a face.

# **Parameters**

#### name

[str] User-defined label for the new surface body.

### face

[Face] Target face to use as the source for the new surface.

### Returns

#### **Body**

Surface body.

#### **Notes**

The source face can be anywhere within the design component hierarchy. Therefore, there is no validation requiring that the face is placed under the target component where the body is to be created.

```
Component.create_body_from_surface(name: str, trimmed_surface:
```

ansys.geometry.core.shapes.surfaces.TrimmedSurface)  $\rightarrow$  ansys.geometry.core.designer.body.Body

Create a surface body from a trimmed surface.

#### **Parameters**

name

[str] User-defined label for the new surface body.

# trimmed\_surface

[TrimmedSurface] Geometry for the new surface body.

#### Returns

**Body** 

Surface body.

#### **Notes**

It is possible to create a closed solid body (as opposed to an open surface body) with a Sphere or Torus if they are untrimmed. This can be validated with *body.is\_surface*.

```
Component.create_surface_from_trimmed_curves(name: str, trimmed_curves:
```

*list[*ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve*]*) → ansys.geometry.core.designer.body.Body

Create a surface body from a list of trimmed curves all lying on the same plane.

#### **Parameters**

name

[str] User-defined label for the new surface body.

# trimmed\_curves

[list[TrimmedCurve]] Curves to define the plane and body.

# Returns

**Body** 

Surface body.

Component.create\_coordinate\_system(name: str, frame: ansys.geometry.core.math.frame.Frame)  $\rightarrow$  ansys.geometry.core.designer.coordinate\_system.CoordinateSystem

Create a coordinate system.

The newly created coordinate system is place under this component within the design assembly.

### **Parameters**

name

[str] User-defined label for the new coordinate system.

# frame

[Frame] Frame defining the coordinate system bounds.

#### Returns

### CoordinateSystem

Translate the bodies in a specified direction by a distance.

#### **Parameters**

bodies: list[Body]

list of bodies to translate by the same distance.

**direction:** UnitVector3D Direction of the translation.

distance: ~pint.Quantity | Distance | Real

Magnitude of the translation.

# Returns

None

#### **Notes**

If the body does not belong to this component (or its children), it is not translated.

```
Component.create_beams (segments: list[tuple[ansys.geometry.core.math.point.Point3D, ansys.geometry.core.math.point.Point3D]], profile: ansys.geometry.core.designer.beam.BeamProfile) \rightarrow list[ansys.geometry.core.designer.beam.Beam]
```

Create beams under the component.

# **Parameters**

# segments

[list[tuple[Point3D, Point3D]]] list of start and end pairs, each specifying a single line segment.

#### profile

[BeamProfile] Beam profile to use to create the beams.

### **Notes**

The newly created beams synchronize to a design within a supporting Geometry service instance.

```
Component.create_beam(start: ansys.geometry.core.math.point.Point3D, end: ansys.geometry.core.math.point.Point3D, profile: ansys.geometry.core.designer.beam.BeamProfile) \rightarrow ansys.geometry.core.designer.beam.Beam
```

Create a beam under the component.

The newly created beam synchronizes to a design within a supporting Geometry service instance.

#### **Parameters**

start

[Point3D] Starting point of the beam line segment.

```
end
                    [Point3D] Ending point of the beam line segment.
                profile
                    [BeamProfile] Beam profile to use to create the beam.
Component.delete_component(component: Component | str) \rightarrow None
      Delete a component (itself or its children).
           Parameters
                component
                    [Component | str] ID of the component or instance to delete.
      Notes
      If the component is not this component (or its children), it is not deleted.
Component.delete_body(body: ansys.geometry.core.designer.body.Body | str) \rightarrow None
      Delete a body belonging to this component (or its children).
           Parameters
                body
                    [Body | str] ID of the body or instance to delete.
      Notes
      If the body does not belong to this component (or its children), it is not deleted.
Component.add_design_point(name: str, point: ansys.geometry.core.math.point.Point3D) →
                                  ansys.geometry.core.designer.designpoint.DesignPoint
      Create a single design point.
           Parameters
                name
                    [str] User-defined label for the design points.
                points
                    [Point3D] 3D point constituting the design point.
Component.add_design_points(name: str, points: list[ansys.geometry.core.math.point.Point3D]) →
                                   list[ansys.geometry.core.designer.designpoint.DesignPoint]
      Create a list of design points.
           Parameters
                name
                    [str] User-defined label for the list of design points.
                points
                    [list[Point3D]] list of the 3D points that constitute the list of design points.
Component.delete_beam(beam: ansys.geometry.core.designer.beam.Beam | str) \rightarrow None
      Delete an existing beam belonging to this component's scope.
           Parameters
                beam
                    [Beam | str] ID of the beam or instance to delete.
```

# **Notes**

If the beam belongs to this component's children, it is deleted. If the beam does not belong to this component (or its children), it is not deleted.

# Component.search\_component(id: str) $\rightarrow Component$ | None

Search nested components recursively for a component.

#### **Parameters**

id

[str] ID of the component to search for.

#### **Returns**

# Component

Component with the requested ID. If this ID is not found, None is returned.

 $\textbf{Component.search\_body}(\textit{id: str}) \rightarrow \textit{ansys.geometry.core.designer.body.Body} \mid \textbf{None}$ 

Search bodies in the component's scope.

#### **Parameters**

id

[str] ID of the body to search for.

#### **Returns**

# Body | None

Body with the requested ID. If the ID is not found, None is returned.

#### **Notes**

This method searches for bodies in the component and nested components recursively.

 $\textbf{Component.search\_beam}(\textit{id: str}) \rightarrow \textit{ansys.geometry.core.designer.beam.Beam} \mid \textbf{None}$ 

Search beams in the component's scope.

# **Parameters**

id

[str] ID of the beam to search for.

#### Returns

### Beam | None

Beam with the requested ID. If the ID is not found, None is returned.

#### **Notes**

This method searches for beams in the component and nested components recursively.

Component.tessellate( recursive call: bool = False)  $\rightarrow$  pyvista.PolyData | list[pyvista.MultiBlock]

Tessellate the component.

# **Parameters**

# \_recursive\_call: bool, default: False

Internal flag to indicate if this method is being called recursively. Not to be used by the user.

#### Returns

### PolyData, list[MultiBlock]

Tessellated component as a single PolyData object. If the method is called recursively, a list of MultiBlock objects is returned.

Component.plot(merge\_component: bool = True, merge\_bodies: bool = True, screenshot: str | None = None, use\_trame: bool | None = None, use\_service\_colors: bool | None = None, allow\_picking: bool | None = None, \*\*plotting\_options: dict | None)  $\rightarrow$  None | list[Any]

Plot the component.

#### **Parameters**

### merge\_component

[bool, default: True] Whether to merge the component into a single dataset. By default, True. Performance improved. When True, all the faces of the component are effectively merged into a single dataset. If False, the individual bodies are kept separate.

#### merge bodies

[bool, default: True] Whether to merge each body into a single dataset. By default, True. Performance improved. When True, all the faces of each individual body are effectively merged into a single dataset. If False, the individual faces are kept separate.

#### screenshot

[str, default: None] Path for saving a screenshot of the image being represented.

#### use trame

[bool, default: None] Whether to enable the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

#### use\_service\_colors

[bool, default: None] Whether to use the colors assigned to the body in the service. The default is None, in which case the ansys.geometry.core.USE\_SERVICE\_COLORS global setting is used.

# allow\_picking

[bool, default: None] Whether to enable picking. The default is None, in which case the picker is not enabled.

# \*\*plotting\_options

[dict, default: None] Keyword arguments for plotting. For allowable keyword arguments, see the

# Returns

# None | list[Any]

If allow\_picking=True, a list of picked objects is returned. Otherwise, None.

# **Examples**

Create 25 small cylinders in a grid-like pattern on the XY plane and plot them. Make the cylinders look metallic by enabling physically-based rendering with pbr=True.

```
>>> from ansys.geometry.core.misc.units import UNITS as u
>>> from ansys.geometry.core.sketch import Sketch
>>> from ansys.geometry.core.math import Plane, Point2D, Point3D, UnitVector3D
>>> from ansys.geometry.core import Modeler
>>> import numpy as np
>>> modeler = Modeler()
>>> origin = Point3D([0, 0, 0])
>>> plane = Plane(origin, direction_x=[1, 0, 0], direction_y=[0, 1, 0])
```

(continues on next page)

(continued from previous page)

```
>>> design = modeler.create_design("my-design")
>>> mycomp = design.add_component("my-comp")
>>> n = 5
>>> xx, yy = np.meshgrid(
        np.linspace(-4, 4, n),
        np.linspace(-4, 4, n),
. . .
...)
>>> for x, y in zip(xx.ravel(), yy.ravel()):
        sketch = Sketch(plane)
        sketch.circle(Point2D([x, y]), 0.2 * u.m)
        mycomp.extrude\_sketch(f"body-{x}-{y}", sketch, 1 * u.m)
>>> mycomp
ansys.geometry.core.designer.Component 0x2203cc9ec50
    Name
                         : my-comp
                         : True
    Exists
   Parent component
                         : my-design
    N Bodies
                         : 25
    N Components
    N Coordinate Systems: 0
>>> mycomp.plot(pbr=True, metallic=1.0)
```

Component.\_\_repr\_\_()  $\rightarrow$  str

Represent the Component as a string.

```
Component.tree_print(consider_comps: bool = True, consider_bodies: bool = True, consider_beams: bool = True, depth: int | None = None, indent: int = 4, sort_keys: bool = False, return_list: bool = False, skip_loc_header: bool = False) \rightarrow None | list[str]
```

Print the component in tree format.

# **Parameters**

```
consider_comps
    [bool, default: True] Whether to print the nested components.

consider_bodies
    [bool, default: True] Whether to print the bodies.

consider_beams
    [bool, default: True] Whether to print the beams.

depth
    [int | None, default: None] Depth level to print. If None, it prints all levels.

indent
    [int, default: 4] Indentation level. Minimum is 2 - if less than 2, it is set to 2 by default.

sort_keys
    [bool, default: False] Whether to sort the keys alphabetically.

return_list
    [bool, default: False] Whether to return a list of strings or print out the tree structure.

skip_loc_header
    [bool, default: False] Whether to skip the location header. Mostly for internal use.
```

7.1. The ansys.geometry.core library

Returns

### None | list[str]

Tree-style printed component or list of strings representing the component tree.

# SharedTopologyType

```
class ansys.geometry.core.designer.component.SharedTopologyType(*args, **kwds)
```

Bases: enum. Enum

Shared topologies available.

# **Overview**

# **Attributes**

SHARETYPE\_NONE
SHARETYPE\_SHARE
SHARETYPE\_MERGE
SHARETYPE\_GROUPS

# Import detail

from ansys.geometry.core.designer.component import SharedTopologyType

# **Attribute detail**

SharedTopologyType.SHARETYPE\_NONE = 0

 $SharedTopologyType.SHARETYPE\_SHARE = 1$ 

SharedTopologyType.SHARETYPE\_MERGE = 2

SharedTopologyType.SHARETYPE\_GROUPS = 3

# ExtrusionDirection

class ansys.geometry.core.designer.component.ExtrusionDirection(\*args, \*\*kwds)

Bases: enum. Enum

Enum for extrusion direction definition.

# **Overview**

# **Constructors**

from\_string Convert a string to an ExtrusionDirection enum.

# **Attributes**

POSITIVE NEGATIVE

# Import detail

from ansys.geometry.core.designer.component import ExtrusionDirection

#### Attribute detail

```
ExtrusionDirection.POSITIVE = '+'
ExtrusionDirection.NEGATIVE = '-'
```

# **Method detail**

classmethod ExtrusionDirection.from\_string( $string: str, use\_default\_if\_error: bool = False$ )  $\rightarrow$  ExtrusionDirection

Convert a string to an ExtrusionDirection enum.

# **Description**

Provides for managing components.

The coordinate\_system.py module

# **Summary**

#### **Classes**

CoordinateSystem Represents a user-defined coordinate system within the design assembly.

# CoordinateSystem

class ansys.geometry.core.designer.coordinate\_system.CoordinateSystem(name: str, frame: an-

sys.geometry.core.math.frame.Frame, parent\_component: ansys.geometry.core.designer.component.Comgrpc\_client: ansys.geometry.core.connection.client.GrpcCl

preexisting\_id: str |
None = None)

Represents a user-defined coordinate system within the design assembly.

This class synchronizes to a design within a supporting Geometry service instance.

# **Parameters**

#### name

[str] User-defined label for the coordinate system.

# frame

[Frame] Frame defining the coordinate system bounds.

# parent\_component

[Component, default: Component] Parent component the coordinate system is assigned against.

### grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

# **Overview**

# **Properties**

id	ID of the coordinate system.
name	Name of the coordinate system.
frame	Frame of the coordinate system.
<pre>parent_component</pre>	Parent component of the coordinate system.
is_alive	Flag indicating if coordinate system is still alive on the server.

# **Special methods**

repr Represent the coordinate system as a strin	g.
-------------------------------------------------	----

# Import detail

from ansys.geometry.core.designer.coordinate\_system import CoordinateSystem

# **Property detail**

property CoordinateSystem.id: str

ID of the coordinate system.

property CoordinateSystem.name: str

Name of the coordinate system.

property CoordinateSystem.frame: ansys.geometry.core.math.frame.Frame

Frame of the coordinate system.

property CoordinateSystem.parent\_component:
 ansys.geometry.core.designer.component.Component

Parent component of the coordinate system.

property CoordinateSystem.is\_alive: bool

Flag indicating if coordinate system is still alive on the server.

# **Method detail**

CoordinateSystem. $\_$ repr $\_$ ()  $\rightarrow$  str

Represent the coordinate system as a string.

# **Description**

Provides for managing a user-defined coordinate system.

# The design.py module

# **Summary**

# **Classes**

Design Provides for organizing geometry assemblies.

#### **Enums**

*DesignFileFormat* Provides supported file formats that can be downloaded for designs.

# Design

**class** ansys.geometry.core.designer.design.**Design**(name: str, modeler:

ansys.geometry.core.modeler.Modeler, read\_existing\_design: bool = False)

Bases: ansys.geometry.core.designer.component.Component

Provides for organizing geometry assemblies.

This class synchronizes to a supporting Geometry service instance.

# **Parameters**

#### name

[str] User-defined label for the design.

# grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

# read\_existing\_design

[bool, default: False] Whether an existing design on the service should be read. This parameter is only valid when connecting to an existing service session. Otherwise, avoid using this optional parameter.

#### Overview

# Methods

Close the design.
Add a material to the design.
Save a design to disk on the active Geometry server instance.
Export and download the design from the server.
Export the design to an scdocx file.
Export the design to an dsco file.
Export the design to an stride file.
Export the design to a Parasolid text file.
Export the design to a Parasolid binary file.
Export the design to an FMD file.
Export the design to a STEP file.
Export the design to an IGES file.
Export the design to a PMDB file.
Create a named selection on the active Geometry server instance.
Delete a named selection on the active Geometry server instance.
Delete a component (itself or its children).
Set the shared topology to apply to the component.
Add a new beam circular profile under the design for creating beams.
Get parameters for the design.
Set or update a parameter of the design.
Add a mid-surface thickness to a list of bodies.
Add a mid-surface offset type to a list of bodies.
Remove a beam profile on the active geometry server instance.
Insert a file into the design.

# **Properties**

design_id	The design's object unique id.
materials	List of materials available for the design.
named_selections	List of named selections available for the design.
beam_profiles	List of beam profile available for the design.
parameters	List of parameters available for the design.
is_active	Whether the design is currently active.
is_closed	Whether the design is closed (i.e. not active).

# Special methods

\_\_repr\_\_ Represent the Design as a string.

# Import detail

```
from ansys.geometry.core.designer.design import Design
```

# **Property detail**

# property Design.design\_id: str

The design's object unique id.

property Design.materials: list[ansys.geometry.core.materials.material.Material]

List of materials available for the design.

property Design.named\_selections:

list[ansys.geometry.core.designer.selection.NamedSelection]

List of named selections available for the design.

property Design.beam\_profiles: list[ansys.geometry.core.designer.beam.BeamProfile]

List of beam profile available for the design.

property Design.parameters: list[ansys.geometry.core.parameters.parameter.Parameter]

List of parameters available for the design.

property Design.is\_active: bool

Whether the design is currently active.

property Design.is\_closed: bool

Whether the design is closed (i.e. not active).

# **Method detail**

Design.close()  $\rightarrow$  None

Close the design.

Design.add\_material(material: ansys.geometry.core.materials.material.Material)  $\rightarrow$  None

Add a material to the design.

### **Parameters**

#### material

[Material] Material to add.

Design.save( $file\_location: pathlib.Path \mid str$ )  $\rightarrow$  None

Save a design to disk on the active Geometry server instance.

# **Parameters**

### file location

[Path | str] Location on disk to save the file to.

 $\label{eq:design-design-design} Design. \textbf{download} (file\_location: pathlib.Path \mid str, format: DesignFileFormat = DesignFileFormat.SCDOCX) \rightarrow None$ 

Export and download the design from the server.

# **Parameters**

# file\_location

[Path | str] Location on disk to save the file to.

#### format

[DesignFileFormat, default: DesignFileFormat.SCDOCX] Format for the file to save to.

Design.export\_to\_scdocx(location: pathlib.Path | str | None = None)  $\rightarrow$  pathlib.Path

Export the design to an scdocx file.

#### **Parameters**

#### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

#### Returns

#### Path

The path to the saved file.

Design.export\_to\_disco(location: pathlib.Path |  $str \mid None = None$ )  $\rightarrow$  pathlib.Path

Export the design to an dsco file.

#### **Parameters**

#### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

#### Returns

#### Path

The path to the saved file.

Design.export\_to\_stride(location: pathlib.Path | str | None = None)  $\rightarrow$  pathlib.Path

Export the design to an stride file.

### **Parameters**

#### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

# Returns

#### Path

The path to the saved file.

Design.export\_to\_parasolid\_text(location: pathlib.Path | str | None = None)  $\rightarrow$  pathlib.Path

Export the design to a Parasolid text file.

# **Parameters**

# location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

# Returns

### Path

The path to the saved file.

 $Design. \textbf{export\_to\_parasolid\_bin}(location: pathlib.Path \mid str \mid None = None) \rightarrow pathlib.Path$ 

Export the design to a Parasolid binary file.

#### **Parameters**

# location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

#### **Returns**

#### Path

The path to the saved file.

Design.export\_to\_fmd(location: pathlib.Path | str | None = None)  $\rightarrow$  pathlib.Path

Export the design to an FMD file.

#### **Parameters**

#### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

#### **Returns**

#### Path

The path to the saved file.

Design.export\_to\_step(location: pathlib.Path |  $str \mid None = None$ )  $\rightarrow$  pathlib.Path

Export the design to a STEP file.

### **Parameters**

#### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

# Returns

### Path

The path to the saved file.

Design.export\_to\_iges(location: pathlib.Path | str = None)  $\rightarrow$  pathlib.Path

Export the design to an IGES file.

# **Parameters**

### location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

#### Returns

# Path

The path to the saved file.

 $Design.export\_to\_pmdb(location: pathlib.Path | str | None = None) \rightarrow pathlib.Path$ 

Export the design to a PMDB file.

#### **Parameters**

# location

[Path | str, optional] Location on disk to save the file to. If None, the file will be saved in the current working directory.

# Returns

#### Path

The path to the saved file.

Design.create\_named\_selection(name: str, bodies: list[ansys.geometry.core.designer.body.Body] | None = None, faces: list[ansys.geometry.core.designer.face.Face] | None = None, edges: list[ansys.geometry.core.designer.edge.Edge] | None = None, beams: list[ansys.geometry.core.designer.beam.Beam] | None = None, design\_points: list[ansys.geometry.core.designer.designpoint.DesignPoint] | None = None) 

-> ansys.geometry.core.designer.selection.NamedSelection

Create a named selection on the active Geometry server instance.

#### **Parameters**

#### name

[str] User-defined name for the named selection.

### **bodies**

[list[Body], default: None] All bodies to include in the named selection.

#### faces

[list[Face], default: None] All faces to include in the named selection.

# edges

[list[Edge], default: None] All edges to include in the named selection.

#### beams

[list[Beam], default: None] All beams to include in the named selection.

# design\_points

[list[DesignPoint], default: None] All design points to include in the named selection.

#### Returns

#### NamedSelection

Newly created named selection that maintains references to all target entities.

Design.delete\_named\_selection(named\_selection: ansys.geometry.core.designer.selection.NamedSelection | str)  $\rightarrow$  None

Delete a named selection on the active Geometry server instance.

# Parameters

### named selection

[NamedSelection | str] Name of the named selection or instance.

Design.delete\_component(component: ansys.geometry.core.designer.component.Component | str)  $\rightarrow$  None Delete a component (itself or its children).

# **Parameters**

id

[Union[Component, str]] Name of the component or instance to delete.

#### Raises

### ValueError

The design itself cannot be deleted.

#### **Notes**

If the component is not this component (or its children), it is not deleted.

 $\textbf{Design.set\_shared\_topology}(\textit{share\_type}: ansys.geometry.core.designer.component.SharedTopologyType) \rightarrow \\ None$ 

Set the shared topology to apply to the component.

#### **Parameters**

#### share type

[SharedTopologyType] Shared topology type to assign.

#### Raises

#### ValueError

Shared topology does not apply to a design.

Design.add\_beam\_circular\_profile(name: str, radius: pint.Quantity |

ansys.geometry.core.misc.measurements.Distance, center:
numpy.ndarray | ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.point.Point3D = ZERO\_POINT3D,
direction\_x: numpy.ndarray | ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_X,
direction\_y: numpy.ndarray | ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_Y) →

ansys.geometry.core.designer.beam.BeamCircularProfile

Add a new beam circular profile under the design for creating beams.

#### **Parameters**

#### name

[str] User-defined label for the new beam circular profile.

#### radius

[Quantity | Distance] Radius of the beam circular profile.

#### center

[ndarray | RealSequence | Point3D] Center of the beam circular profile.

#### direction\_x

[ndarray | RealSequence | UnitVector3D | Vector3D] X-plane direction.

#### direction v

[ndarray | RealSequence | UnitVector3D | Vector3D] Y-plane direction.

 $\label{eq:design_get_all_parameters} Design. \mbox{get\_all\_parameters}() \rightarrow list[\mbox{\it ansys.geometry.core.parameters.parameter.Parameter}]$  Get parameters for the design.

# Returns

### list[Parameter]

List of parameters for the design.

Design.set\_parameter(dimension: ansys.geometry.core.parameters.parameter.Parameter)  $\rightarrow$  ansys.geometry.core.parameters.parameter.ParameterUpdateStatus

Set or update a parameter of the design.

# **Parameters**

# dimension

[Parameter] Parameter to set.

#### Returns

# ParameterUpdateStatus

Status of the update operation.

```
Design.add_midsurface_thickness(thickness: pint.Quantity, bodies: list[ansys.geometry.core.designer.body.Body]) \rightarrow None
```

Add a mid-surface thickness to a list of bodies.

#### **Parameters**

#### thickness

[Quantity] Thickness to be assigned.

#### **bodies**

[list[Body]] All bodies to include in the mid-surface thickness assignment.

# **Notes**

Only surface bodies will be eligible for mid-surface thickness assignment.

Design.add\_midsurface\_offset(offset\_type: ansys.geometry.core.designer.body.MidSurfaceOffsetType, bodies:  $list[ansys.geometry.core.designer.body.Body]) \rightarrow None$ 

Add a mid-surface offset type to a list of bodies.

### **Parameters**

#### offset type

[MidSurfaceOffsetType] Surface offset to be assigned.

#### **bodies**

[list[Body]] All bodies to include in the mid-surface offset assignment.

#### **Notes**

Only surface bodies will be eligible for mid-surface offset assignment.

Design.delete\_beam\_profile(beam\_profile: ansys.geometry.core.designer.beam.BeamProfile | str)  $\rightarrow$  None Remove a beam profile on the active geometry server instance.

### **Parameters**

# beam\_profile

[BeamProfile | str] A beam profile name or instance that should be deleted.

```
Design.insert_file(file_location: pathlib.Path | str, import_options:
```

ansys.geometry.core.misc.options.ImportOptions = ImportOptions())  $\rightarrow$  ansys.geometry.core.designer.component.Component

Insert a file into the design.

# **Parameters**

### file location

[Path | str] Location on disk where the file is located.

#### import options

[ImportOptions] The options to pass into upload file

# Returns

# Component

The newly inserted component.

```
\texttt{Design.\_\_repr}\_\_() \to \mathsf{str}
```

Represent the Design as a string.

# DesignFileFormat

class ansys.geometry.core.designer.design.DesignFileFormat(\*args, \*\*kwds)

Bases: enum. Enum

Provides supported file formats that can be downloaded for designs.

# Overview

# **Attributes**

SCDOCX	
PARASOLID_TEXT	
PARASOLID_BIN	
FMD	
STEP	
IGES	
PMDB	
STRIDE	
DISCO	
INVALID	

# Import detail

from ansys.geometry.core.designer.design import DesignFileFormat

# **Attribute detail**

DesignFileFormat.SCDOCX

DesignFileFormat.PARASOLID\_TEXT

DesignFileFormat.PARASOLID\_BIN

DesignFileFormat.**FMD** 

DesignFileFormat.STEP

DesignFileFormat. IGES

DesignFileFormat.PMDB

 ${\tt DesignFileFormat.} \textbf{STRIDE}$ 

 ${\tt DesignFileFormat.} \textbf{DISCO}$ 

DesignFileFormat.INVALID = ('INVALID', None)

# **Description**

Provides for managing designs.

# The designpoint.py module

# **Summary**

# **Classes**

DesignPoint Provides for creating design points in components.

# DesignPoint

class ansys.geometry.core.designer.designpoint.DesignPoint(id: str, name: str, point: an-

sys.geometry.core.math.point.Point3D,

parent\_component: an-

sys.geometry.core.designer.component.Component)

Provides for creating design points in components.

#### **Parameters**

id

[str] Server-defined ID for the design points.

#### name

[str] User-defined label for the design points.

# points

[Point3D] 3D point constituting the design points.

# parent\_component

[Component] Parent component to place the new design point under within the design assembly.

#### Overview

# **Properties**

id	ID of the design point.
name	Name of the design point.
value	Value of the design point.
<pre>parent_component</pre>	Component node that the design point is under.

# **Special methods**

\_\_repr\_\_ Represent the design points as a string.

# Import detail

from ansys.geometry.core.designer.designpoint import DesignPoint

# **Property detail**

```
property DesignPoint.id: str
    ID of the design point.

property DesignPoint.name: str
    Name of the design point.

property DesignPoint.value: ansys.geometry.core.math.point.Point3D
    Value of the design point.

property DesignPoint.parent_component: ansys.geometry.core.designer.component.Component
    Component node that the design point is under.
```

#### Method detail

```
DesignPoint.__repr__() \rightarrow str
Represent the design points as a string.
```

# **Description**

Module for creating and managing design points.

# The edge.py module

# **Summary**

### **Classes**

*Edge* Represents a single edge of a body within the design assembly.

#### **Enums**

CurveType Provides values for the curve types supported.

# Edge

```
class ansys.geometry.core.designer.edge.Edge(id: str, curve\_type: CurveType, body: ansys.geometry.core.designer.body.Body, <math>grpc\_client: ansys.geometry.core.connection.client.GrpcClient, <math>is\_reversed: bool = False)
```

Represents a single edge of a body within the design assembly.

This class synchronizes to a design within a supporting Geometry service instance.

# **Parameters**

```
id
    [str] Server-defined ID for the body.
curve_type
    [CurveType] Type of curve that the edge forms.
body
    [Body] Parent body that the edge constructs.
```

# grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

# is\_reversed

[bool] Direction of the edge.

#### Overview

# **Properties**

id	ID of the edge.
body	Body of the edge.
is_reversed	Flag indicating if the edge is reversed.
shape	Underlying trimmed curve of the edge.
length	Calculated length of the edge.
curve_type	Curve type of the edge.
faces	Faces that contain the edge.
start	Start point of the edge.
end	End point of the edge.

# Import detail

```
from ansys.geometry.core.designer.edge import Edge
```

# **Property detail**

property Edge.body: ansys.geometry.core.designer.body.Body

Body of the edge.

property Edge.is\_reversed: bool
 Flag indicating if the edge is reversed.

property Edge.shape: ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve

Underlying trimmed curve of the edge.

If the edge is reversed, its shape is the ReversedTrimmedCurve type, which swaps the start and end points of the curve and handles parameters to allow evaluation as if the curve is not reversed.

```
property Edge.length: pint.Quantity
```

Calculated length of the edge.

property Edge.curve\_type: CurveType

Curve type of the edge.

property Edge.faces: list[ansys.geometry.core.designer.face.Face]

Faces that contain the edge.

property Edge.start: ansys.geometry.core.math.point.Point3D

Start point of the edge.

property Edge.end: ansys.geometry.core.math.point.Point3D

End point of the edge.

# CurveType

class ansys.geometry.core.designer.edge.CurveType(\*args, \*\*kwds)

Bases: enum. Enum

Provides values for the curve types supported.

# Overview

# **Attributes**

CURVETYPE\_UNKNOWN
CURVETYPE\_LINE
CURVETYPE\_CIRCLE
CURVETYPE\_ELLIPSE
CURVETYPE\_NURBS
CURVETYPE\_PROCEDURAL

# Import detail

from ansys.geometry.core.designer.edge import CurveType

# **Attribute detail**

CurveType.CURVETYPE\_UNKNOWN = 0

CurveType.CURVETYPE\_LINE = 1

CurveType.CURVETYPE\_CIRCLE = 2

CurveType.CURVETYPE\_ELLIPSE = 3

CurveType.CURVETYPE\_NURBS = 4

CurveType.CURVETYPE\_PROCEDURAL = 5

# Description

Module for managing an edge.

# The face.py module

# **Summary**

# **Classes**

FaceLoop	Provides an internal class holding the face loops defined.
Face	Represents a single face of a body within the design assembly.

# **Enums**

SurfaceType	Provides values for the surface types supported.
FaceLoopType	Provides values for the face loop types supported.

# FaceLoop

Provides an internal class holding the face loops defined.

#### **Parameters**

```
type
    [FaceLoopType] Type of loop.
length
    [Quantity] Length of the loop.
min_bbox
    [Point3D] Minimum point of the bounding box containing the loop.
max_bbox
    [Point3D] Maximum point of the bounding box containing the loop.
edges
    [list[Edge]] Edges contained in the loop.
```

# **Notes**

This class is to be used only when parsing server side results. It is not intended to be instantiated by a user.

# **Overview**

# **Properties**

type	Type of the loop.
length	Length of the loop.
min_bbox	Minimum point of the bounding box containing the loop.
max_bbox	Maximum point of the bounding box containing the loop.
edges	Edges contained in the loop.

# Import detail

```
from ansys.geometry.core.designer.face import FaceLoop
```

# **Property detail**

```
property FaceLoop.length: pint.Quantity
```

Length of the loop.

property FaceLoop.min\_bbox: ansys.geometry.core.math.point.Point3D

Minimum point of the bounding box containing the loop.

property FaceLoop.max\_bbox: ansys.geometry.core.math.point.Point3D

Maximum point of the bounding box containing the loop.

property FaceLoop.edges: list[ansys.geometry.core.designer.edge.Edge]

Edges contained in the loop.

#### Face

class ansys.geometry.core.designer.face.Face( $id: str, surface\_type:$  SurfaceType, body: ansys.geometry.core.designer.body.Body,  $grpc\_client:$  ansys.geometry.core.connection.client.GrpcClient,  $is\_reversed: bool = False$ )

Represents a single face of a body within the design assembly.

This class synchronizes to a design within a supporting Geometry service instance.

### **Parameters**

id

[str] Server-defined ID for the body.

### surface\_type

[SurfaceType] Type of surface that the face forms.

#### body

[Body] Parent body that the face constructs.

# grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

# Overview

# **Methods**

set_color	Set the color of the face.
normal	Get the normal direction to the face at certain UV coordinates.
face_normal	Get the normal direction to the face at certain UV coordinates.
point	Get a point of the face evaluated at certain UV coordinates.
face_point	Get a point of the face evaluated at certain UV coordinates.
create_isoparametric_curves	Create isoparametic curves at the given proportional parameter.
setup_offset_relationship	Create an offset relationship between two faces.
tessellate	•
	Tessellate the face and return the geometry as triangles.
plot	Plot the face.

# **Properties**

id	Face ID.
is_reversed	Flag indicating if the face is reversed.
body	Body that the face belongs to.
shape	Underlying trimmed surface of the face.
surface_type	Surface type of the face.
area	Calculated area of the face.
edges	List of all edges of the face.
loops	List of all loops of the face.
color	Get the current color of the face.
bounding_box	Get the bounding box for the face.

# Import detail

```
from ansys.geometry.core.designer.face import Face
```

# **Property detail**

property Face.id: str

Face ID.

property Face.is\_reversed: bool

Flag indicating if the face is reversed.

property Face.body: ansys.geometry.core.designer.body.Body

Body that the face belongs to.

property Face.shape: ansys.geometry.core.shapes.surfaces.trimmed\_surface.TrimmedSurface

Underlying trimmed surface of the face.

If the face is reversed, its shape is a ReversedTrimmedSurface type, which handles the direction of the normal vector to ensure it is always facing outward.

property Face.surface\_type: SurfaceType

Surface type of the face.

property Face.area: pint.Quantity

Calculated area of the face.

property Face.edges: list[ansys.geometry.core.designer.edge.Edge]

List of all edges of the face.

property Face.loops: list[FaceLoop]

List of all loops of the face. property Face.color: str

Get the current color of the face.

property Face.bounding\_box: ansys.geometry.core.math.bbox.BoundingBox2D

Get the bounding box for the face.

# **Method detail**

Face.**set\_color**(color:  $str \mid tuple[float, float, float]) <math>\rightarrow$  None

Set the color of the face.

Face.normal(u: float = 0.5, v: float = 0.5)  $\rightarrow$  ansys.geometry.core.math.vector.UnitVector3D

Get the normal direction to the face at certain UV coordinates.

#### **Parameters**

u

[float, default: 0.5] First coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

 $\mathbf{v}$ 

[float, default: 0.5] Second coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

#### Returns

#### UnitVector3D

UnitVector3D object evaluated at the given U and V coordinates. This UnitVector3D object is perpendicular to the surface at the given UV coordinates.

#### **Notes**

To properly use this method, you must handle UV coordinates. Thus, you must know how these relate to the underlying Geometry service. It is an advanced method for Geometry experts only.

Face. **face\_normal** (u: float = 0.5, v: float = 0.5)  $\rightarrow$  ansys.geometry.core.math.vector.UnitVector3D

Get the normal direction to the face at certain UV coordinates.

# **Parameters**

11

[float, default: 0.5] First coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

v

[float, default: 0.5] Second coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

### Returns

#### UnitVector3D

UnitVector3D object evaluated at the given U and V coordinates. This UnitVector3D object is perpendicular to the surface at the given UV coordinates.

#### **Notes**

This method is deprecated. Use the normal method instead.

Face.point(u: float = 0.5, v: float = 0.5)  $\rightarrow$  ansys.geometry.core.math.point.Point3D

Get a point of the face evaluated at certain UV coordinates.

# **Parameters**

11

[float, default: 0.5] First coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

[float, default: 0.5] Second coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

#### Returns

#### Point3D

Point 3D object evaluated at the given UV coordinates.

# **Notes**

To properly use this method, you must handle UV coordinates. Thus, you must know how these relate to the underlying Geometry service. It is an advanced method for Geometry experts only.

Face.  $face\_point(u: float = 0.5, v: float = 0.5) \rightarrow ansys.geometry.core.math.point.Point3D$ 

Get a point of the face evaluated at certain UV coordinates.

# **Parameters**

u [float, default: 0.5] First coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

[float, default: 0.5] Second coordinate of the 2D representation of a surface in UV space. The default is 0.5, which is the center of the surface.

#### Returns

#### Point3D

Point 3D object evaluated at the given UV coordinates.

### **Notes**

This method is deprecated. Use the point method instead.

```
Face.create_isoparametric_curves(use_u_param: bool, parameter: float) \rightarrow list[ansys.geometry.core.shapes.curves.trimmed_curve.TrimmedCurve]
```

Create isoparametic curves at the given proportional parameter.

Typically, only one curve is created, but if the face has a hole, it is possible that more than one curve is created.

#### **Parameters**

# use\_u\_param

[bool] Whether the parameter is the u coordinate or v coordinate. If True, it is the u coordinate. If False, it is the v coordinate.

# parameter

[float] Proportional [0-1] parameter to create the one or more curves at.

### Returns

# list[TrimmedCurve]

list of curves that were created.

Face. setup\_offset\_relationship(other\_face: Face, set\_baselines: bool = False, process\_adjacent\_faces: bool = False)  $\rightarrow$  bool

Create an offset relationship between two faces.

# **Parameters**

#### other face

[Face] The face to setup an offset relationship with.

#### set baselines

[bool, default: False] Automatically set baseline faces.

# process\_adjacent\_faces

[bool, default: False] Look for relationships of the same offset on adjacent faces.

#### Returns

bool

True when successful, False when failed.

### Face.tessellate() $\rightarrow$ pyvista.PolyData

Tessellate the face and return the geometry as triangles.

#### Returns

#### PolvData

pyvista.PolyData object holding the face.

Face.plot(screenshot:  $str \mid None = None$ ,  $use\_trame$ :  $bool \mid None = None$ ,  $use\_service\_colors$ :  $bool \mid None = None$ , \*\*plotting\_options:  $dict \mid None$ )  $\rightarrow$  None

Plot the face.

#### **Parameters**

#### screenshot

[str, default: None] Path for saving a screenshot of the image that is being represented.

#### use trame

[bool, default: None] Whether to enable the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

#### use\_service\_colors

[bool, default: None] Whether to use the colors assigned to the face in the service. The default is None, in which case the ansys.geometry.core.USE\_SERVICE\_COLORS global setting is used.

# \*\*plotting\_options

[dict, default: None] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

#### SurfaceType

class ansys.geometry.core.designer.face.SurfaceType(\*args, \*\*kwds)

Bases: enum. Enum

Provides values for the surface types supported.

### Overview

# **Attributes**

SURFACETYPE\_UNKNOWN
SURFACETYPE\_PLANE
SURFACETYPE\_CYLINDER
SURFACETYPE\_CONE
SURFACETYPE\_TORUS
SURFACETYPE\_SPHERE
SURFACETYPE\_NURBS
SURFACETYPE\_PROCEDURAL

# Import detail

from ansys.geometry.core.designer.face import SurfaceType

# **Attribute detail**

```
SurfaceType.SURFACETYPE_UNKNOWN = 0
SurfaceType.SURFACETYPE_PLANE = 1
SurfaceType.SURFACETYPE_CYLINDER = 2
SurfaceType.SURFACETYPE_CONE = 3
SurfaceType.SURFACETYPE_TORUS = 4
SurfaceType.SURFACETYPE_SPHERE = 5
SurfaceType.SURFACETYPE_NURBS = 6
SurfaceType.SURFACETYPE_PROCEDURAL = 7
```

# FaceLoopType

Provides values for the face loop types supported.

# **Overview**

# **Attributes**

INNER\_LOOP OUTER\_LOOP

# Import detail

from ansys.geometry.core.designer.face import FaceLoopType

# **Attribute detail**

FaceLoopType.INNER\_LOOP = 'INNER'
FaceLoopType.OUTER\_LOOP = 'OUTER'

# **Description**

Module for managing a face.

The geometry\_commands.py module

# **Summary**

# **Classes**

GeometryCommands	Provides geometry commands for PyAnsys Geometry.

# **Enums**

ExtrudeType	Provides values for extrusion types.
OffsetMode	Provides values for offset modes during extrusions.
FillPatternType	Provides values for types of fill patterns.

# GeometryCommands

class ansys.geometry.core.designer.geometry\_commands.GeometryCommands(grpc\_client: an-

sys.geometry.core.connection.client.GrpcCl

Provides geometry commands for PyAnsys Geometry.

# **Parameters**

grpc\_client

[GrpcClient] gRPC client to use for the geometry commands.

# **Overview**

## Methods

chamfer	Create a chamfer on an edge or adjust the chamfer of a face.
fillet	Create a fillet on an edge or adjust the fillet of a face.
full_fillet	Create a full fillet betweens a collection of faces.
extrude_faces	Extrude a selection of faces.
<pre>extrude_faces_up_to</pre>	Extrude a selection of faces up to another object.
extrude_edges	Extrude a selection of edges. Provide either a face or a direction and point.
extrude_edges_up_to	Extrude a selection of edges up to another object.
rename_object	Rename an object.
create_linear_pattern	Create a linear pattern. The pattern can be one or two dimensions.
<pre>modify_linear_pattern</pre>	Modify a linear pattern. Leave an argument at 0 for it to remain unchanged.
create_circular_pattern	Create a circular pattern. The pattern can be one or two dimensions.
<pre>modify_circular_pattern</pre>	Modify a circular pattern. Leave an argument at 0 for it to remain unchanged.
create_fill_pattern	Create a fill pattern.
update_fill_pattern	Update a fill pattern.
revolve_faces	Revolve face around an axis.
revolve_faces_up_to	Revolve face around an axis up to a certain object.
revolve_faces_by_helix	Revolve face around an axis in a helix shape.
replace_face	Replace a face with another face.
split_body	Split bodies with a plane, slicers, or faces.
get_round_info	Get info on the rounding of a face.
offset_faces_set_radius	Offset faces with a radius.

## Import detail

from ansys.geometry.core.designer.geometry\_commands import GeometryCommands

list[ansys.geometry.core.designer.edge.Edge] |

GeometryCommands.chamfer(selection: ansys.geometry.core.designer.edge.Edge |

### **Method detail**

```
ansys.geometry.core.designer.face.Face |
list[ansys.geometry.core.designer.face.Face], distance:
ansys.geometry.core.typing.Real) → bool

Create a chamfer on an edge or adjust the chamfer of a face.

Parameters
selection
[Edge | list[Edge] | Face | list[Face]] One or more edges or faces to act on.
distance
[Real] Chamfer distance.

Returns
bool
True when successful, False when failed.

GeometryCommands.fillet(selection: ansys.geometry.core.designer.edge.Edge |
```

list[ansys.geometry.core.designer.edge.Edge] |
ansys.geometry.core.designer.face.Face |

*ansys.geometry.core.typing.Real*)  $\rightarrow$  bool

list[ansys.geometry.core.designer.face.Face], radius:

Create a fillet on an edge or adjust the fillet of a face.

#### **Parameters**

#### selection

[Edge | list[Edge] | Face | list[Face]] One or more edges or faces to act on.

#### radius

[Real] Fillet radius.

### Returns

bool

True when successful, False when failed.

GeometryCommands.full\_fillet( $faces: list[ansys.geometry.core.designer.face.Face]) <math>\rightarrow bool$ 

Create a full fillet betweens a collection of faces.

### **Parameters**

#### faces

[list[Face]] Faces to round.

#### Returns

bool

True when successful, False when failed.

```
GeometryCommands.extrude_faces(faces: ansys.geometry.core.designer.face.Face |
```

list[ansys.geometry.core.designer.face.Face], distance:

ansys.geometry.core.typing.Real, direction:

ansys.geometry.core.math.vector.UnitVector3D = *None*, *extrude\_type*:

ExtrudeType = ExtrudeType.ADD, offset\_mode: OffsetMode =

 $OffsetMode.MOVE\_FACES\_TOGETHER, pull\_symmetric:\ bool = False,$ 

copy: bool = False,  $force\_do\_as\_extrude$ : bool = False)  $\rightarrow$ 

list[ansys.geometry.core.designer.body.Body]

Extrude a selection of faces.

#### **Parameters**

#### faces

[Face | list[Face]] Faces to extrude.

## distance

[Real] Distance to extrude.

#### direction

[UnitVector3D, default: None] Direction of extrusion. If no direction is provided, it will be inferred.

### extrude\_type

[ExtrudeType, default: ExtrudeType.ADD] Type of extrusion to be performed.

### offset mode

 $[ {\tt OffsetMode}, default: {\tt \it OffsetMode}. {\tt \it MOVE\_FACES\_TOGETHER} ] \ Mode of how to handle offset relationships.$ 

### pull\_symmetric

[bool, default: False] Pull symmetrically on both sides if True.

### copy

[bool, default: False] Copy the face and move it instead of extruding the original face if True.

### force do as extrude

[bool, default: False] Forces to do as an extrusion if True, if False allows extrusion by offset.

#### Returns

# list[Body]

Bodies created by the extrusion if any.

## GeometryCommands.extrude\_faces\_up\_to(faces: ansys.geometry.core.designer.face.Face |

list[ansys.geometry.core.designer.face.Face], up\_to\_selection:
ansys.geometry.core.designer.face.Face |
ansys.geometry.core.designer.edge.Edge |
ansys.geometry.core.designer.body.Body, seed\_point:
ansys.geometry.core.math.point.Point3D, direction:
ansys.geometry.core.math.vector.UnitVector3D, extrude\_type:
ExtrudeType = ExtrudeType.ADD, offset\_mode: OffsetMode =
OffsetMode.MOVE\_FACES\_TOGETHER, pull\_symmetric: bool =
False, copy: bool = False, force\_do\_as\_extrude: bool = False) →
list[ansys.geometry.core.designer.body.Body]

Extrude a selection of faces up to another object.

#### **Parameters**

#### faces

[Face | list[Face]] Faces to extrude.

#### up to selection

[Face | Edge | Body] The object to pull the faces up to.

## seed\_point

[Point3D] Origin to define the extrusion.

#### direction

[UnitVector3D, default: None] Direction of extrusion. If no direction is provided, it will be inferred.

### extrude\_type

[ExtrudeType, default: ExtrudeType.ADD] Type of extrusion to be performed.

#### offset mode

 $[Offset \verb|Mode|, default|: \textit{OffsetMode.MOVE\_FACES\_TOGETHER}] \ Mode of how to handle offset relationships.$ 

### pull\_symmetric

[bool, default: False] Pull symmetrically on both sides if True.

#### copy

[bool, default: False] Copy the face and move it instead of extruding the original face if True.

#### force do as extrude

[bool, default: False] Forces to do as an extrusion if True, if False allows extrusion by offset.

## Returns

## list[Body]

Bodies created by the extrusion if any.

GeometryCommands.extrude\_edges(edges: ansys.geometry.core.designer.edge.Edge | list[ansys.geometry.core.designer.edge.Edge], distance: ansys.geometry.core.typing.Real, from\_face: ansys.geometry.core.designer.face.Face = None, from\_point: ansys.geometry.core.math.point.Point3D = None, direction: ansys.geometry.core.math.vector.UnitVector3D = None, extrude\_type: ExtrudeType = ExtrudeType.ADD, pull\_symmetric: bool = False, copy: bool = False, natural\_extension: bool = False)  $\rightarrow$ 

list[ansys.geometry.core.designer.body.Body]

Extrude a selection of edges. Provide either a face or a direction and point.

### **Parameters**

#### edges

[Edge | list[Edge]] Edges to extrude.

#### distance

[Real] Distance to extrude.

#### from face

[Face, default: None] Face to pull normal from.

#### from point

[Point3D, default: None] Point to pull from. Must be used with direction.

#### direction

[UnitVector3D, default: None] Direction to pull. Must be used with from\_point.

#### extrude type

[ExtrudeType, default: ExtrudeType.ADD] Type of extrusion to be performed.

#### pull\_symmetric

[bool, default: False] Pull symmetrically on both sides if True.

### copy

[bool, default: False] Copy the edge and move it instead of extruding the original edge if True.

## natural\_extension

[bool, default: False] Surfaces will extend in a natural or linear shape after exceeding its original range.

### Returns

#### list[Body]

Bodies created by the extrusion if any.

```
GeometryCommands.extrude_edges_up_to(edges: ansys.geometry.core.designer.edge.Edge |
```

list[ansys.geometry.core.designer.edge.Edge], up\_to\_selection: ansys.geometry.core.designer.face.Face | ansys.geometry.core.designer.edge.Edge | ansys.geometry.core.designer.body.Body, seed\_point: ansys.geometry.core.math.point.Point3D, direction: ansys.geometry.core.math.vector.UnitVector3D, extrude\_type: ExtrudeType = ExtrudeType.ADD) → list[ansys.geometry.core.designer.body.Body]

Extrude a selection of edges up to another object.

```
Parameters
               edges
                    [Edge | list[Edge]] Edges to extrude.
               up to selection
                    [Face, default: None] The object to pull the faces up to.
               seed point
                    [Point3D] Origin to define the extrusion.
               direction
                    [UnitVector3D, default: None] Direction of extrusion.
               extrude type
                    [ExtrudeType, default: ExtrudeType.ADD] Type of extrusion to be performed.
           Returns
               list[Body]
                    Bodies created by the extrusion if any.
GeometryCommands.rename_object(selection: list[ansys.geometry.core.designer.body.Body] |
                                       list[ ansys.geometry.core.designer.component.Component ]
                                       list[ansys.geometry.core.designer.face.Face] |
                                       list[ansys.geometry.core.designer.edge.Edge], name: str) → bool
      Rename an object.
           Parameters
               selection
                   [list[Body] | list[Component] | list[Face] | list[Edge]] Selection of the object to re-
                   name.
               name
                    [str] New name for the object.
           Returns
               bool
                   True when successful, False when failed.
GeometryCommands.create_linear_pattern(selection: ansys.geometry.core.designer.face.Face |
                                                 list/ansys.geometry.core.designer.face.Face], linear_direction:
                                                 ansys.geometry.core.designer.edge.Edge |
                                                 ansys.geometry.core.designer.face.Face, count_x: int, pitch_x:
                                                 ansys.geometry.core.typing.Real, two_dimensional: bool = False,
                                                 count\_y:\ int = None,\ pitch\_y:\ ansys.geometry.core.typing.Real =
                                                 None) \rightarrow bool
      Create a linear pattern. The pattern can be one or two dimensions.
           Parameters
               selection
                    [Face | list[Face]] Faces to create the pattern out of.
               linear direction
                    [Edge | Face] Direction of the linear pattern, determined by the direction of an edge or face
                    normal.
               count x
                    [int] How many times the pattern repeats in the x direction.
```

# pitch x [Real] The spacing between each pattern member in the x direction. two dimensional [bool, default: False] If True, create a pattern in the x and y direction. count y [int, default: None] How many times the pattern repeats in the y direction. pitch y [Real, default: None] The spacing between each pattern member in the y direction. Returns bool True when successful, False when failed. GeometryCommands.modify\_linear\_pattern(selection: ansys.geometry.core.designer.face.Face | $list[ansys.geometry.core.designer.face.Face], count\_x: int = 0,$ $pitch_x$ : $ansys.geometry.core.typing.Real = 0.0, count_y$ : int = 0, pitch v: ansys.geometry.core.typing.Real = 0.0, new seed index: int = 0, $old\_seed\_index$ : int = 0) $\rightarrow$ bool Modify a linear pattern. Leave an argument at 0 for it to remain unchanged. **Parameters** selection [Face | list[Face]] Faces that belong to the pattern. count x [int, default: 0] How many times the pattern repeats in the x direction. pitch x [Real, default: 0.0] The spacing between each pattern member in the x direction. [int, default: 0] How many times the pattern repeats in the y direction. pitch v [Real, default: 0.0] The spacing between each pattern member in the y direction. new\_seed\_index [int, default: 0] The new seed index of the member. old seed index [int, default: 0] The old seed index of the member. Returns bool True when successful, False when failed. GeometryCommands.create\_circular\_pattern(selection: ansys.geometry.core.designer.face.Face | list/ansys.geometry.core.designer.face.Face/, circular axis: ansys.geometry.core.designer.edge.Edge, circular\_count: int, circular\_angle: ansys.geometry.core.typing.Real, two dimensional: bool = False, linear count: int = None, *linear\_pitch: ansys.geometry.core.typing.Real = None*, radial\_direction: ansys.geometry.core.math.vector.UnitVector3D = None) $\rightarrow$ bool

Create a circular pattern. The pattern can be one or two dimensions.

#### **Parameters**

#### selection

[Face | list[Face]] Faces to create the pattern out of.

#### circular axis

[Edge] The axis of the circular pattern, determined by the direction of an edge.

### circular count

[int] How many members are in the circular pattern.

### circular\_angle

[Real] The angular range of the pattern.

#### two dimensional

[bool, default: False] If True, create a two-dimensional pattern.

### linear\_count

[int, default: None] How many times the circular pattern repeats along the radial lines for a two-dimensional pattern.

### linear\_pitch

[Real, default: None] The spacing along the radial lines for a two-dimensional pattern.

## radial direction

[UnitVector3D, default: None] The direction from the center out for a two-dimensional pattern.

#### Returns

#### bool

True when successful, False when failed.

GeometryCommands.modify\_circular\_pattern(selection: ansys.geometry.core.designer.face.Face |

list[ansys.geometry.core.designer.face.Face], circular\_count: int = 0, linear\_count: int = 0, step\_angle: ansys.geometry.core.typing.Real = 0.0, step\_linear: ansys.geometry.core.typing.Real = 0.0)  $\rightarrow$  bool

Modify a circular pattern. Leave an argument at 0 for it to remain unchanged.

## **Parameters**

## selection

[Face | list[Face]] Faces that belong to the pattern.

## circular\_count

[int, default: 0] How many members are in the circular pattern.

#### linear count

[int, default: 0] How many times the circular pattern repeats along the radial lines for a two-dimensional pattern.

## step\_angle

[Real, default: 0.0] Defines the circular angle.

# step\_linear

[Real, default: 0.0] Defines the step, along the radial lines, for a pattern dimension greater than 1.

## Returns

#### bool

True when successful, False when failed.

## GeometryCommands.create\_fill\_pattern(selection: ansys.geometry.core.designer.face.Face |

list[ansys.geometry.core.designer.face.Face], linear\_direction:
ansys.geometry.core.designer.edge.Edge |
ansys.geometry.core.designer.face.Face, fill\_pattern\_type:
FillPatternType, margin: ansys.geometry.core.typing.Real,
x\_spacing: ansys.geometry.core.typing.Real, y\_spacing:
ansys.geometry.core.typing.Real, row\_x\_offset:
ansys.geometry.core.typing.Real = 0, row\_y\_offset:
ansys.geometry.core.typing.Real = 0, column\_x\_offset:
ansys.geometry.core.typing.Real = 0, column\_y\_offset:
ansys.geometry.core.typing.Real = 0 → bool

Create a fill pattern.

### **Parameters**

#### selection

[Face | list[Face]] Faces to create the pattern out of.

### linear direction

[Edge] Direction of the linear pattern, determined by the direction of an edge.

## fill\_pattern\_type

[FillPatternType] The type of fill pattern.

### margin

[Real] Margin defining the border of the fill pattern.

### x\_spacing

[Real] Spacing between the pattern members in the x direction.

### y\_spacing

[Real] Spacing between the pattern members in the x direction.

#### row\_x\_offset

[Real, default: 0] Offset for the rows in the x direction. Only used with FillPattern. SKEWED.

### row\_y\_offset

[Real, default: 0] Offset for the rows in the y direction. Only used with FillPattern. SKEWED.

### column\_x\_offset

[Real, default: 0] Offset for the columns in the x direction. Only used with FillPattern. SKEWED.

## column\_y\_offset

[Real, default: 0] Offset for the columns in the y direction. Only used with FillPattern. SKEWED.

### Returns

## bool

True when successful, False when failed.

GeometryCommands.update\_fill\_pattern(selection: ansys.geometry.core.designer.face.Face |  $list[ansys.geometry.core.designer.face.Face]) \rightarrow bool$ 

Update a fill pattern.

When the face that a fill pattern exists upon changes in size, the fill pattern can be updated to fill the new space.

#### **Parameters**

```
selection
                    [Face | list[Face]] Face(s) that are part of a fill pattern.
           Returns
                bool
                    True when successful, False when failed.
GeometryCommands.revolve_faces(selection: ansys.geometry.core.designer.face.Face |
                                        list/ansys.geometry.core.designer.face.Face/, axis:
                                        ansys.geometry.core.shapes.curves.line.Line, angle:
                                        ansys.geometry.core.typing.Real) \rightarrow
                                        list[ansys.geometry.core.designer.body.Body]
      Revolve face around an axis.
           Parameters
                selection
                    [Face | list[Face]] Face(s) to revolve.
                axis
                    [Line] Axis of revolution.
                angle
                    [Real] Angular distance to revolve.
           Returns
                list[Body]
                    Bodies created by the extrusion if any.
GeometryCommands.revolve_faces_up_to(selection: ansys.geometry.core.designer.face.Face |
                                               list[ansys.geometry.core.designer.face.Face], up_to:
                                               ansys.geometry.core.designer.face.Face |
                                               ansys.geometry.core.designer.edge.Edge |
                                               ansys.geometry.core.designer.body.Body, axis:
                                               ansys.geometry.core.shapes.curves.line.Line, direction:
                                               ansys.geometry.core.math.vector.UnitVector 3D, {\it extrude\_type:}
                                               ExtrudeType = ExtrudeType.ADD) \rightarrow
                                               list[ansys.geometry.core.designer.body.Body]
      Revolve face around an axis up to a certain object.
           Parameters
                selection
                    [Face | list[Face]] Face(s) to revolve.
                up_to
                    [Face | Edge | Body] Object to revolve the face up to.
                axis
                    [Line] Axis of revolution.
                direction
                    [UnitVector3D] Direction of extrusion.
                extrude_type
                    [ExtrudeType, default: ExtrudeType.ADD] Type of extrusion to be performed.
           Returns
```

```
list[Body]
Bodies commetryCommands.revol
```

Bodies created by the extrusion if any.

GeometryCommands.revolve\_faces\_by\_helix(selection: ansys.geometry.core.designer.face.Face | list[ansys.geometry.core.designer.face.Face], axis: ansys.geometry.core.shapes.curves.line.Line, direction: ansys.geometry.core.math.vector.UnitVector3D, height: ansys.geometry.core.typing.Real, pitch: ansys.geometry.core.typing.Real, taper\_angle: ansys.geometry.core.typing.Real, right\_handed: bool, both\_sides: bool)  $\rightarrow$ 

Revolve face around an axis in a helix shape.

## **Parameters**

### selection

[Face | list[Face]] Face(s) to revolve.

axis

[Line] Axis of revolution.

#### direction

[UnitVector3D] Direction of extrusion.

### height

[Real,] Height of the helix.

### pitch

[Real,] Pitch of the helix.

### taper\_angle

[Real,] Tape angle of the helix.

## right\_handed

[bool,] Right-handed helix if True, left-handed if False.

#### both sides

[bool,] Create on both sides if True, one side if False.

### Returns

## list[Body]

Bodies created by the extrusion if any.

GeometryCommands.replace\_face(target\_selection: ansys.geometry.core.designer.face.Face | list[ansys.geometry.core.designer.face.Face], replacement\_selection:

ansys.geometry.core.designer.face.Face | ansys.geometry.core.designer.face.Face

list[ansys.geometry.core.designer.body.Body]

 $list[ansys.geometry.core.designer.face.Face]) \rightarrow bool$ 

Replace a face with another face.

### **Parameters**

#### target\_selection

[Union[Face, list[Face]]] The face or faces to replace.

# replacement\_selection

[Union[Face, list[Face]]] The face or faces to replace with.

#### Returns

```
bool
                    True when successful. False when failed.
GeometryCommands.split_body(bodies: list[ansys.geometry.core.designer.body.Body], plane:
                                   ansys.geometry.core.math.plane.Plane, slicers:
                                   ansys.geometry.core.designer.edge.Edge |
                                   list/ansys.geometry.core.designer.edge.Edge/
                                   ansys.geometry.core.designer.face.Face |
                                   list/ansys.geometry.core.designer.face.Face/, faces:
                                   list[ansys.geometry.core.designer.face.Face], extendfaces: bool) \rightarrow bool
      Split bodies with a plane, slicers, or faces.
           Parameters
               bodies
                    [list[Body]] Bodies to split.
               plane
                    [Plane] Plane to split with
               slicers
                    [Edge | list[Edge] | Face | list[Face]] Slicers to split with.
               faces
                    [list[Face]] Faces to split with.
               extendFaces
                    [bool] Extend faces if split with faces.
           Returns
               bool
                    True when successful, False when failed.
GeometryCommands.get_round_info(face: ansys.geometry.core.designer.face.Face) \rightarrow tuple[bool,
                                        ansys.geometry.core.typing.Real]
      Get info on the rounding of a face.
           Parameters
               Face
                    The design face to get round info on.
           Returns
               tuple[bool, Real]
                    True if round is aligned with face's U-parameter direction, False otherwise. Radius of the
                    round.
GeometryCommands.offset_faces_set_radius(faces: ansys.geometry.core.designer.face.Face |
                                                    list[ansys.geometry.core.designer.face.Face], radius:
                                                    ansys.geometry.core.typing.Real, copy: bool = False,
                                                    offset mode: OffsetMode =
                                                    OffsetMode.IGNORE_RELATIONSHIPS, extrude_type:
                                                    ExtrudeType = ExtrudeType.FORCE\_INDEPENDENT) \rightarrow
                                                    bool
      Offset faces with a radius.
           Parameters
               faces
                    [Face | list[Face]] Faces to offset.
```

#### radius

[Real] Radius of the offset.

#### copy

[bool, default: False] Copy the face and move it instead of offsetting the original face if True.

#### offset mode

[OffsetMode, default: OffsetMode.MOVE\_FACES\_TOGETHER] Mode of how to handle offset relationships.

### extrude\_type

[ExtrudeType, default: ExtrudeType.FORCE\_INDEPENDENT] Type of extrusion to be performed.

#### Returns

#### bool

True when successful, False when failed.

### ExtrudeType

class ansys.geometry.core.designer.geometry\_commands.ExtrudeType(\*args, \*\*kwds)

Bases: enum. Enum

Provides values for extrusion types.

#### Overview

# **Attributes**

NONE
ADD
CUT
FORCE\_ADD
FORCE\_CUT
FORCE\_INDEPENDENT
FORCE\_NEW\_SURFACE

# Import detail

from ansys.geometry.core.designer.geometry\_commands import ExtrudeType

## Attribute detail

```
ExtrudeType.NONE = 0
ExtrudeType.ADD = 1
ExtrudeType.CUT = 2
ExtrudeType.FORCE_ADD = 3
ExtrudeType.FORCE_CUT = 4
```

```
ExtrudeType.FORCE_INDEPENDENT = 5
ExtrudeType.FORCE_NEW_SURFACE = 6
```

### OffsetMode

class ansys.geometry.core.designer.geometry\_commands.OffsetMode(\*args, \*\*kwds)

Bases: enum. Enum

Provides values for offset modes during extrusions.

### Overview

### **Attributes**

IGNORE\_RELATIONSHIPS
MOVE\_FACES\_TOGETHER
MOVE\_FACES\_APART

## Import detail

from ansys.geometry.core.designer.geometry\_commands import OffsetMode

## **Attribute detail**

```
OffsetMode.IGNORE_RELATIONSHIPS = 0
OffsetMode.MOVE_FACES_TOGETHER = 1
OffsetMode.MOVE_FACES_APART = 2
```

## FillPatternType

 $\textbf{class} \ \, \text{ansys.geometry.core.designer.geometry\_commands.} \\ \textbf{FillPatternType}(*\textit{args}, **kwds)$ 

Bases: enum.Enum

Provides values for types of fill patterns.

### Overview

### **Attributes**

GRID OFFSET SKEWED

## Import detail

from ansys.geometry.core.designer.geometry\_commands import FillPatternType

## **Attribute detail**

FillPatternType.GRID = 0
FillPatternType.OFFSET = 1
FillPatternType.SKEWED = 2

## **Description**

Provides tools for pulling geometry.

# The part.py module

# **Summary**

## Classes

Part	Represents a part master.
MasterComponent	Represents a part occurrence.

#### Part

list[ansys.geometry.core.designer.body.MasterBody])

Represents a part master.

This class should not be accessed by users. The Part class holds fundamental data of an assembly.

## **Parameters**

id

[str] Unique identifier for the part.

#### name

[str] Name of the part.

# components

[list[MasterComponent]] list of MasterComponent children that the part contains.

#### bodies

[list[MasterBody]] list of MasterBody children that the part contains. These are master bodies.

#### Overview

# **Properties**

id	ID of the part.
name	Name of the part.
components	MasterComponent children that the part contains.
bodies	MasterBody children that the part contains.

### Special methods

\_\_repr\_\_ Represent the part as a string.

# Import detail

```
from ansys.geometry.core.designer.part import Part
```

# **Property detail**

```
property Part.id: str
ID of the part.

property Part.name: str
Name of the part.

property Part.components: list[MasterComponent]
MasterComponent children that the part contains.

property Part.bodies: list[ansys.geometry.core.designer.body.MasterBody]
MasterBody children that the part contains.

These are master bodies.
```

## **Method detail**

```
Part.__repr__() \rightarrow str
Represent the part as a string.
```

## MasterComponent

Represents a part occurrence.

#### **Parameters**

```
id
    [str] Unique identifier for the transformed part.

name
    [str] Name of the transformed part.

part
    [Part] Reference to the transformed part's master part.

transform
    [Matrix44] 4x4 transformation matrix from the master part.
```

### **Notes**

This class should not be accessed by users. It holds the fundamental data of an assembly. Master components wrap parts by adding a transform matrix.

## **Overview**

## **Properties**

id	ID of the transformed part.
name	Name of the transformed part.
occurrences	List of all occurrences of the component.
part	Master part of the transformed part.
transform	4x4 transformation matrix from the master part.

# **Special methods**

\_\_repr\_\_ Represent the master component as a string.

## Import detail

from ansys.geometry.core.designer.part import MasterComponent

## **Property detail**

property MasterComponent.id: str

ID of the transformed part.

property MasterComponent.name: str

Name of the transformed part.

property MasterComponent.occurrences:

list[ansys.geometry.core.designer.component.Component]

List of all occurrences of the component.

property MasterComponent.part: Part

Master part of the transformed part.

## **Method detail**

MasterComponent.\_\_repr\_\_()  $\rightarrow$  str

Represent the master component as a string.

# **Description**

Module providing fundamental data of an assembly.

## The selection.py module

# **Summary**

## **Classes**

 ${\it NamedSelection} \quad \text{Represents a single named selection within the design assembly}.$ 

#### NamedSelection

```
class ansys.geometry.core.designer.selection.NamedSelection(name: str, grpc_client: an-
```

sys.geometry.core.connection.client.GrpcClient, *bodies:* 

list[ansys.geometry.core.designer.body.Body]

| *None* = *None*, *faces*:

list[ansys.geometry.core.designer.face.Face]

| *None* = *None*, *edges*:

list[ansys.geometry.core.designer.edge.Edge]

| *None* = *None*, beams:

list[ansys.geometry.core.designer.beam.Beam]

| *None* = *None*,  $design\_points$ :

list[ansys.geometry.core.designer.designpoint.DesignPoi

| None = None, preexisting\_id: str |

None = None)

Represents a single named selection within the design assembly.

This class synchronizes to a design within a supporting Geometry service instance.

A named selection organizes one or more design entities together for common actions against the entire group.

#### **Parameters**

#### name

[str] User-defined name for the named selection.

### grpc\_client

[GrpcClient] Active supporting Geometry service instance for design modeling.

#### **bodies**

[list[Body], default: None] All bodies to include in the named selection.

#### faces

[list[Face], default: None] All faces to include in the named selection.

## edges

[list[Edge], default: None] All edges to include in the named selection.

#### beams

[list[Beam], default: None] All beams to include in the named selection.

### design\_points

[list[DesignPoints], default: None] All design points to include in the named selection.

## **Overview**

## **Properties**

id	ID of the named selection.
name	Name of the named selection.

## Import detail

from ansys.geometry.core.designer.selection import NamedSelection

# **Property detail**

property NamedSelection.id: str

ID of the named selection.

property NamedSelection.name: str

Name of the named selection.

# **Description**

Module for creating a named selection.

## **Description**

PyAnsys Geometry designer subpackage.

## The materials package

## **Summary**

### **Submodules**

material	Provides the data structure for material and material properties.
property	Provides the MaterialProperty class.

## The material.py module

# **Summary**

#### **Classes**

Material Provides the data structure for a material.

## **Material**

Provides the data structure for a material.

## **Parameters**

name: str

Material name.

**density: ~pint.Quantity**Material density.

 $additional\_properties:\ Sequence [Material Property],\ default:\ None$ 

Additional material properties.

### **Overview**

### **Methods**

add_property	Add a material property to the Material class.
--------------	------------------------------------------------

## **Properties**

properties	Dictionary of the material property type and material properties.
name	Material name.

# Import detail

```
from ansys.geometry.core.materials.material import Material
```

## **Property detail**

```
property Material.properties:
dict[ansys.geometry.core.materials.property.MaterialPropertyType,
ansys.geometry.core.materials.property.MaterialProperty]
```

Dictionary of the material property type and material properties.

```
property Material.name: str
     Material name.
```

## **Method detail**

```
Material.add_property(type: ansys.geometry.core.materials.property.MaterialPropertyType, name: str,
                            quantity: pint.Quantity) \rightarrow None
```

Add a material property to the Material class.

## **Parameters**

```
type
```

[MaterialPropertyType] Material property type.

name: str

Material name.

# quantity: ~pint.Quantity

Material value and unit.

## **Description**

Provides the data structure for material and material properties.

## The property.py module

## **Summary**

#### **Classes**

MaterialProperty Provides the data structure for a material property.

### **Enums**

MaterialPropertyType Enum holding the possible values for MaterialProperty objects.

## **MaterialProperty**

Provides the data structure for a material property.

### **Parameters**

#### type

[MaterialPropertyType | str] Type of the material property. If the type is a string, it must be a valid material property type - though it might not be supported by the Material-PropertyType enum.

#### name: str

Material property name.

## quantity: ~pint.Quantity | Real

Value and unit in case of a supported Quantity. If the type is not supported, it must be a Real value (float or integer).

### **Overview**

## **Properties**

type	Material property ID.
name	Material property name.
quantity	Material property quantity and unit.

### Import detail

from ansys.geometry.core.materials.property import MaterialProperty

## **Property detail**

property MaterialProperty.type: MaterialPropertyType | str

Material property ID.

If the type is not supported, it will be a string.

property MaterialProperty.name: str

Material property name.

property Material Property.quantity: pint.Quantity | ansys.geometry.core.typing.Real

Material property quantity and unit.

If the type is not supported, it will be a Real value (float or integer).

### MaterialPropertyType

```
class ansys.geometry.core.materials.property.MaterialPropertyType(*args, **kwds)
```

Bases: enum. Enum

Enum holding the possible values for MaterialProperty objects.

#### Overview

#### **Attributes**

DENSITY
ELASTIC\_MODULUS
POISSON\_RATIO
SHEAR\_MODULUS
SPECIFIC\_HEAT
TENSILE\_STRENGTH
THERMAL\_CONDUCTIVITY

# Static methods

from\_id Return the MaterialPropertyType value from the service.

## Import detail

from ansys.geometry.core.materials.property import MaterialPropertyType

## **Attribute detail**

```
MaterialPropertyType.DENSITY = 'Density'
MaterialPropertyType.ELASTIC_MODULUS = 'ElasticModulus'
MaterialPropertyType.POISSON_RATIO = 'PoissonsRatio'
MaterialPropertyType.SHEAR_MODULUS = 'ShearModulus'
MaterialPropertyType.SPECIFIC_HEAT = 'SpecificHeat'
MaterialPropertyType.TENSILE_STRENGTH = 'TensileStrength'
MaterialPropertyType.THERMAL_CONDUCTIVITY = 'ThermalConductivity'
```

### Method detail

```
static MaterialPropertyType.from_id(id: str) \rightarrow MaterialPropertyType Return the MaterialPropertyType value from the service.
```

#### **Parameters**

id

[str] Geometry Service string representation of a property type.

# Returns

## MaterialPropertyType

Common name for property type.

# **Description**

Provides the Material Property class.

## **Description**

PyAnsys Geometry materials subpackage.

## The math package

# **Summary**

### **Submodules**

bbox	Provides for managing a bounding box.
constants	Provides mathematical constants.
frame	Provides for managing a frame.
matrix	Provides matrix primitive representations.
misc	Provides auxiliary math functions for PyAnsys Geometry.
plane	Provides primitive representation of a 2D plane in 3D space.
point	Provides geometry primitive representation for 2D and 3D points.
vector	Provides for creating and managing 2D and 3D vectors.

# The bbox.py module

## **Summary**

# **Classes**

 $\label{eq:boundingBox2D} \textit{Maintains the } X \textit{ and } Y \textit{ dimensions.}$ 

### BoundingBox2D

 ${\bf class} \ \, {\bf ansys.geometry.core.math.bbox.BoundingBox2D} (x\_min: ansys.geometry.core.typing.Real = sys.float\_info.max, x\_max:$ 

ansys.geometry.core.typing.Real =
sys.float\_info.min, y\_min:
ansys.geometry.core.typing.Real =
sys.float\_info.max, y\_max:
ansys.geometry.core.typing.Real =
sys.float\_info.min)

Maintains the X and Y dimensions.

### **Parameters**

#### x min

[Real] Minimum value for the x-dimensional bounds.

# x\_max

[Real] Maximum value for the x-dimensional bounds.

#### y\_min

[Real] Minimum value for the y-dimensional bounds.

# y\_max

[Real] Maximum value for the y-dimensional bounds.

#### Overview

### **Methods**

add_point	Extend the ranges of the bounding box to include a point.
add_point_components	Extend the ranges of the bounding box to include the X and Y values.
add_points	Extend the ranges of the bounding box to include given points.
contains_point	Evaluate whether a point lies within the X and Y range bounds.
contains_point_components	Check if point components are within the X and Y range bounds.

# **Properties**

x_min	Minimum value of X-dimensional bounds.
x_max	Maximum value of the X-dimensional bounds.
y_min	Minimum value of Y-dimensional bounds.
y_max	Maximum value of Y-dimensional bounds.

## Static methods

<pre>intersect_bboxes Find the interse</pre>	ction of 2 BoundingBox2D objects.
----------------------------------------------	-----------------------------------

# **Special methods**

eq	Equals operator for the BoundingBox2D class.
ne	Not equals operator for the BoundingBox2D class.

# Import detail

from ansys.geometry.core.math.bbox import BoundingBox2D

# **Property detail**

property BoundingBox2D.x\_min: ansys.geometry.core.typing.Real

Minimum value of X-dimensional bounds.

#### **Returns**

# Real

Minimum value of the X-dimensional bounds.

property BoundingBox2D.x\_max: ansys.geometry.core.typing.Real

Maximum value of the X-dimensional bounds.

#### Returns

### Real

Maximum value of the X-dimensional bounds.

#### property BoundingBox2D.y\_min: ansys.geometry.core.typing.Real

Minimum value of Y-dimensional bounds.

### Returns

#### Real

Minimum value of Y-dimensional bounds.

# property BoundingBox2D.y\_max: ansys.geometry.core.typing.Real

Maximum value of Y-dimensional bounds.

### Returns

#### Real

Maximum value of Y-dimensional bounds.

#### Method detail

 $BoundingBox2D. \textbf{add\_point}(\textit{point}: ansys.geometry.core.math.point.Point2D) \rightarrow None$ 

Extend the ranges of the bounding box to include a point.

#### **Parameters**

### point

[Point2D] Point to include within the bounds.

#### **Notes**

This method is only applicable if the point components are outside the current bounds.

BoundingBox2D.add\_point\_components(x: ansys.geometry.core.typing.Real, y: ansys.geometry.core.typing.Real)  $\rightarrow$  None

Extend the ranges of the bounding box to include the X and Y values.

### **Parameters**

X

[Real] Point X component to include within the bounds.

y

[Real] Point Y component to include within the bounds.

## **Notes**

This method is only applicable if the point components are outside the current bounds.

 $BoundingBox2D. \textbf{add\_points}(points: list[ansys.geometry.core.math.point.Point2D]) \rightarrow None$ 

Extend the ranges of the bounding box to include given points.

#### **Parameters**

### points

[list[Point2D]] List of points to include within the bounds.

 $BoundingBox2D. \textbf{contains\_point}(point: ansys.geometry.core.math.point.Point2D) \rightarrow bool$ 

Evaluate whether a point lies within the X and Y range bounds.

#### **Parameters**

#### point

[Point2D] Point to compare against the bounds.

#### Returns

bool

True if the point is contained in the bounding box. Otherwise, False.

BoundingBox2D.contains\_point\_components(x: ansys.geometry.core.typing.Real, y: ansys.geometry.core.typing.Real)  $\rightarrow$  bool

Check if point components are within the X and Y range bounds.

#### **Parameters**

x [Real] Point X component to compare against the bounds.

y
[Real] Point Y component to compare against the bounds.

#### Returns

bool

True if the components are contained in the bounding box. Otherwise, False.

BoundingBox2D. $\_$ eq $\_$ (other: BoundingBox2D)  $\rightarrow$  bool

Equals operator for the BoundingBox2D class.

BoundingBox2D.\_\_ne\_\_(other: BoundingBox2D)  $\rightarrow$  bool

Not equals operator for the BoundingBox2D class.

 $\textbf{static} \ \ \textbf{BoundingBox2D.intersect\_bboxes} (box\_1: \ \textbf{BoundingBox2D}, box\_2: \ \textbf{BoundingBox2D}) \rightarrow \textbf{None} \mid \\ BoundingBox2D$ 

Find the intersection of 2 BoundingBox2D objects.

## **Parameters**

## box\_1: BoundingBox2D

The box to consider the intersection of with respect to box\_2.

# box\_2: BoundingBox2D

The box to consider the intersection of with respect to box\_1.

#### Returns

# **BoundingBox2D:**

The box representing the intersection of the two passed in boxes.

## **Description**

Provides for managing a bounding box.

# The constants.py module

# **Summary**

## **Constants**

DEFAULT_POINT3D	Default value for a 3D point.
DEFAULT_POINT2D	Default value for a 2D point.
IDENTITY_MATRIX33	Identity for a Matrix33 object.
IDENTITY_MATRIX44	Identity for a Matrix44 object.
UNITVECTOR3D_X	Default 3D unit vector in the Cartesian traditional X direction.
UNITVECTOR3D_Y	Default 3D unit vector in the Cartesian traditional Y direction.
UNITVECTOR3D_Z	Default 3D unit vector in the Cartesian traditional Z direction.
UNITVECTOR2D_X	Default 2D unit vector in the Cartesian traditional X direction.
UNITVECTOR2D_Y	Default 2D unit vector in the Cartesian traditional Y direction.
ZERO_VECTOR3D	Zero-valued Vector3D object.
ZERO_VECTOR2D	Zero-valued Vector2D object.
ZERO_POINT3D	Zero-valued Point3D object.
ZERO_POINT2D	Zero-valued Point2D object.

# **Description**

Provides mathematical constants.

### Module detail

### constants.DEFAULT\_POINT3D

Default value for a 3D point.

## constants.DEFAULT\_POINT2D

Default value for a 2D point.

## constants. IDENTITY\_MATRIX33

Identity for a Matrix33 object.

## constants. IDENTITY\_MATRIX44

Identity for a Matrix44 object.

## constants.UNITVECTOR3D\_X

Default 3D unit vector in the Cartesian traditional X direction.

## constants.UNITVECTOR3D\_Y

Default 3D unit vector in the Cartesian traditional Y direction.

### constants.UNITVECTOR3D\_Z

Default 3D unit vector in the Cartesian traditional Z direction.

## constants.UNITVECTOR2D\_X

Default 2D unit vector in the Cartesian traditional X direction.

### constants.UNITVECTOR2D\_Y

Default 2D unit vector in the Cartesian traditional Y direction.

### constants.ZERO\_VECTOR3D

Zero-valued Vector3D object.

# constants.ZERO\_VECTOR2D

Zero-valued Vector2D object.

### constants.ZERO\_POINT3D

Zero-valued Point3D object.

#### constants.ZERO\_POINT2D

Zero-valued Point2D object.

#### The frame.py module

### **Summary**

### **Classes**

Frame Representation of a frame.

### Frame

class ansys.geometry.core.math.frame.Frame(origin: numpy.ndarray |

ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.point.Point3D =
ZERO\_POINT3D, direction\_x: numpy.ndarray |
ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D =
UNITVECTOR3D\_X, direction\_y: numpy.ndarray |
ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D =
UNITVECTOR3D\_Y)

Representation of a frame.

#### **Parameters**

#### origin

[ndarray | RealSequence | Point3D, default: ZERO\_POINT3D] Centered origin of the frame. The default is ZERO\_POINT3D, which is the Cartesian origin.

## direction\_x

[ndarray | RealSequence | UnitVector3D | Vector3D, default: UNITVECTOR3D\_X] X-axis direction.

#### direction\_y

[ndarray | RealSequence | UnitVector3D | Vector3D, default: UNITVECTOR3D\_Y] Yaxis direction.

## **Overview**

### **Methods**

transform\_point2d\_local\_to\_global Transform a 2D point to a global 3D point.

## **Properties**

origin	Origin of the frame.
direction_x	X-axis direction of the frame.
direction_y	Y-axis direction of the frame.
direction_z	Z-axis direction of the frame.
<pre>global_to_local_rotation</pre>	Global to local space transformation matrix.
<pre>local_to_global_rotation</pre>	Local to global space transformation matrix.
transformation_matrix	Full 4x4 transformation matrix.

## **Special methods**

eq	Equals operator for the Frame class.
ne	Not equals operator for the Frame class.

### Import detail

from ansys.geometry.core.math.frame import Frame

# **Property detail**

property Frame.origin: ansys.geometry.core.math.point.Point3D
 Origin of the frame.

property Frame.direction\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-axis direction of the frame.

property Frame.direction\_y: ansys.geometry.core.math.vector.UnitVector3D Y-axis direction of the frame.

property Frame.direction\_z: ansys.geometry.core.math.vector.UnitVector3D
 Z-axis direction of the frame.

#### Returns

#### Matrix33

3x3 matrix representing the transformation from global to local coordinate space, excluding origin translation.

property Frame.local\_to\_global\_rotation: ansys.geometry.core.math.matrix.Matrix33
 Local to global space transformation matrix.

### Returns

## Matrix33

3x3 matrix representing the transformation from local to global coordinate space.

property Frame.transformation\_matrix: ansys.geometry.core.math.matrix.Matrix44
Full 4x4 transformation matrix.

## Returns

#### Matrix44

4x4 matrix representing the transformation from global to local coordinate space.

### **Method detail**

 $\label{point:math:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:point:poin$ 

Transform a 2D point to a global 3D point.

This method transforms a local, plane-contained Point2D object in the global coordinate system, thus representing it as a Point3D object.

#### **Parameters**

#### point

[Point2D] Point2D local object to express in global coordinates.

#### Returns

#### Point3D

Global coordinates for the 3D point.

Frame.\_\_eq\_\_(other: Frame)  $\rightarrow$  bool

Equals operator for the Frame class.

Frame.\_\_ne\_\_(other: Frame)  $\rightarrow$  bool

Not equals operator for the Frame class.

## **Description**

Provides for managing a frame.

## The matrix.py module

## **Summary**

### **Classes**

Matrix	Provides matrix representation.
Matrix33	Provides 3x3 matrix representation.
<i>Matrix44</i>	Provides 4x4 matrix representation.

### **Constants**

DEFAULT_MATRIX33	Default value of the 3x3 identity matrix for the Matrix33 class.
DEFAULT_MATRIX44	Default value of the 4x4 identity matrix for the Matrix44 class.

#### Matrix

Bases: numpy.ndarray

Provides matrix representation.

#### **Parameters**

#### input

[ndarray | RealSequence] Matrix arguments as a np.ndarray class.

#### Overview

#### Methods

determinant	Get the determinant of the matrix.
inverse	Provide the inverse of the matrix.

# **Special methods**

mul	Get the multiplication of the matrix.
eq	Equals operator for the Matrix class.
ne	Not equals operator for the Matrix class.

# Import detail

```
from ansys.geometry.core.math.matrix import Matrix
```

## **Method detail**

```
Matrix.determinant() \rightarrow ansys.geometry.core.typing.Real Get the determinant of the matrix.
```

 $Matrix.inverse() \rightarrow Matrix$ 

Provide the inverse of the matrix.

 $Matrix.\__mul\_\_(other: Matrix | numpy.ndarray) \rightarrow Matrix$ 

Get the multiplication of the matrix.

 $Matrix.\__eq\_(other: Matrix) \rightarrow bool$ 

Equals operator for the Matrix class.

 $Matrix.\__ne\__(other: Matrix) \rightarrow bool$ 

Not equals operator for the Matrix class.

### Matrix33

Bases: Matrix

Provides 3x3 matrix representation.

## **Parameters**

### input

[ndarray | RealSequence | Matrix, default: DEFAULT\_MATRIX33] Matrix arguments as a np.ndarray class.

# Import detail

from ansys.geometry.core.math.matrix import Matrix33

### Matrix44

Bases: Matrix

Provides 4x4 matrix representation.

#### **Parameters**

#### input

[ndarray | RealSequence | Matrix, default: DEFAULT\_MATRIX44] Matrix arguments as a np.ndarray class.

#### Overview

#### **Constructors**

create_translation	Create a matrix representing the specified translation.
create_rotation	Create a matrix representing the specified rotation.

### Methods

 $is\_translation \quad \hbox{Check if the matrix represents a translation}.$ 

# Import detail

from ansys.geometry.core.math.matrix import Matrix44

#### Method detail

 $\textbf{classmethod} \ \texttt{Matrix44.create\_translation} (\textit{translation:} \ \texttt{ansys.geometry.core.math.vector.Vector3D}) \rightarrow \textit{Matrix44}$ 

Create a matrix representing the specified translation.

#### **Parameters**

#### translation

[Vector3D] The translation vector representing the translation. The components of the vector should be in meters.

# Returns

#### Matrix44

A 4x4 matrix representing the translation.

# **Examples**

```
>>> translation_vector = Vector3D(1.0, 2.0, 3.0)
>>> translation_matrix = Matrix44.create_translation(translation_vector)
>>> print(translation_matrix)
[[1. 0. 0. 1.]
[0. 1. 0. 2.]
[0. 0. 1. 3.]
[0. 0. 0. 1.]]
```

# $Matrix44.is\_translation(including\_identity: bool = False) \rightarrow bool$

Check if the matrix represents a translation.

This method checks if the matrix represents a translation transformation. A translation matrix has the following form:

```
[1\ 0\ 0\ tx]\ [0\ 1\ 0\ ty]\ [0\ 0\ 1\ tz]\ [0\ 0\ 0\ 1]
```

#### **Parameters**

### including\_identity

[bool, optional] If True, the method will return True for the identity matrix as well. If False, the method will return False for the identity matrix.

#### Returns

bool

True if the matrix represents a translation, False otherwise.

## **Examples**

classmethod Matrix44.create\_rotation(direction\_x: ansys.geometry.core.math.vector.Vector3D,

direction\_y: ansys.geometry.core.math.vector.Vector3D, direction\_z: ansys.geometry.core.math.vector.Vector3D = None)  $\rightarrow Matrix44$ 

Create a matrix representing the specified rotation.

### **Parameters**

## direction\_x

[Vector3D] The X direction vector.

## direction\_y

[Vector3D] The Y direction vector.

#### direction z

[Vector3D, optional] The Z direction vector. If not provided, it will be calculated as the cross product of direction\_x and direction\_y.

### Returns

#### Matrix44

A 4x4 matrix representing the rotation.

## **Examples**

```
>>> direction_x = Vector3D(1.0, 0.0, 0.0)
>>> direction_y = Vector3D(0.0, 1.0, 0.0)
>>> rotation_matrix = Matrix44.create_rotation(direction_x, direction_y)
>>> print(rotation_matrix)
[[1. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 1. 0.]
[0. 0. 0. 1.]]
```

## **Description**

Provides matrix primitive representations.

#### Module detail

### matrix.DEFAULT\_MATRIX33

Default value of the 3x3 identity matrix for the Matrix33 class.

### matrix.DEFAULT\_MATRIX44

Default value of the 4x4 identity matrix for the Matrix44 class.

## The misc.py module

# **Summary**

### **Functions**

<pre>get_two_circle_intersections</pre>	Get the intersection points of two circles.	
intersect_interval	Find the intersection of two intervals.	

# **Description**

Provides auxiliary math functions for PyAnsys Geometry.

### Module detail

Get the intersection points of two circles.

#### **Parameters**

```
x0
[Rea1] x coordinate of the first circle.
y0
[Rea1] y coordinate of the first circle.
r0
[Rea1] Radius of the first circle.
x1
[Rea1] x coordinate of the second circle.
y1
[Rea1] y coordinate of the second circle.
r1
[Rea1] Radius of the second circle.
```

#### **Returns**

# tuple[tuple[Real, Real], tuple[Real, Real]] | None

Intersection points of the two circles if they intersect. The points are returned as ((x3, y3), (x4, y4)), where (x3, y3) and (x4, y4) are the intersection points of the two circles. If the circles do not intersect, then None is returned.

#### **Notes**

This function is based on the following StackOverflow post: https://stackoverflow.com/questions/55816902/finding-the-intersection-of-two-circles

That post is based on the following implementation: https://paulbourke.net/geometry/circlesphere/

 $\label{eq:misc.intersect_interval} \textit{(first\_min, second\_min, first\_max, second\_max)} \rightarrow \textit{tuple[bool, ansys.geometry.core.typing.Real, ansys.geometry.core.typing.Real]}$ 

Find the intersection of two intervals.

#### **Parameters**

#### first min

[Real] The minimum value of the first interval.

#### second\_min

[Real] The minimum value of the second interval.

### first max

[Real] The maximum value of the first interval.

#### second max

[Real] The maximum value of the second interval.

## Returns

## tuple[bool, Real, Real]

Tuple with a boolean to indicate whether the intervals intersect, the minimum value of the intersection interval, and the maximum value of the intersection interval. If they do not intersect, then the boolean is False and the other values are 0.

# The plane.py module

## Summary

#### **Classes**

*Plane* Provides primitive representation of a 2D plane in 3D space.

### Plane

class ansys.geometry.core.math.plane.Plane(origin: numpy.ndarray |

ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.point.Point3D =
ZERO\_POINT3D, direction\_x: numpy.ndarray |
ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D =
UNITVECTOR3D\_X, direction\_y: numpy.ndarray |
ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.UnitVector3D |
ansys.geometry.core.math.vector.Vector3D =
UNITVECTOR3D\_Y)

Bases: ansys.geometry.core.math.frame.Frame

Provides primitive representation of a 2D plane in 3D space.

### **Parameters**

#### origin

[ndarray | RealSequence | Point3D, default: ZERO\_POINT3D] Centered origin of the frame. The default is ZERO\_POINT3D, which is the Cartesian origin.

### direction x

[ndarray | RealSequence | UnitVector3D | Vector3D, default: UNITVECTOR3D\_X] X-axis direction.

### direction\_y

[ndarray | RealSequence | UnitVector3D | Vector3D, default: UNITVECTOR3D\_Y] Yaxis direction.

### Overview

## **Methods**

*is\_point\_contained* Check if a 3D point is contained in the plane.

### **Properties**

*normal* Calculate the normal vector of the plane.

# **Special methods**

eq	Equals operator for the Plane class.
ne	Not equals operator for the Plane class.

## Import detail

```
from ansys.geometry.core.math.plane import Plane
```

## **Property detail**

property Plane.normal: ansys.geometry.core.math.vector.UnitVector3D

Calculate the normal vector of the plane.

## Returns

#### UnitVector3D

Normal vector of the plane.

### **Method detail**

 ${\tt Plane.is\_point\_contained(\it point: ansys.geometry.core.math.point.Point3D) \rightarrow bool}$ 

Check if a 3D point is contained in the plane.

## **Parameters**

#### point

[Point3D] Point3D class to check.

## Returns

bool

True if the 3D point is contained in the plane, False otherwise.

```
Plane.__eq__(other: Plane) \rightarrow bool
```

Equals operator for the Plane class.

Plane.\_\_ne\_\_(other: Plane)  $\rightarrow$  bool

Not equals operator for the Plane class.

## **Description**

Provides primitive representation of a 2D plane in 3D space.

# The point.py module

## **Summary**

## **Classes**

Point2D	Provides geometry primitive representation for a 2D point.
Point3D	Provides geometry primitive representation for a 3D point.

# **Constants**

DEFAULT_POINT2D_VALUES	Default values for a 2D point.
DEFAULT_POINT3D_VALUES	Default values for a 3D point.
BASE_UNIT_LENGTH	Default value for the length of the base unit.

# Point2D

class ansys.geometry.core.math.point.Point2D(input: numpy.ndarray |

ansys.geometry.core.typing.RealSequence =
DEFAULT\_POINT2D\_VALUES, unit: pint.Unit | None =
None)

Bases: numpy.ndarray, ansys.geometry.core.misc.units.PhysicalQuantity

Provides geometry primitive representation for a 2D point.

#### **Parameters**

#### input

[ndarray | RealSequence, default: DEFAULT\_POINT2D\_VALUES] Direction arguments, either as a numpy.ndarray class or as a RealSequence.

#### unit

[Unit | None, default: DEFAULT\_UNITS.LENGTH] Units for defining 2D point values. If not specified, the default unit is DEFAULT\_UNITS.LENGTH.

#### Overview

### Methods

unit	Get the unit of the object.
base_unit	Get the base unit of the object.

### **Properties**

X X plane component value.Y plane component value.

#### **Attributes**

flat

# **Special methods**

eq	Equals operator for the Point2D class.
ne	Not equals operator for the Point2D class.
add	Add operation for the Point2D class.
sub	Subtraction operation for the Point2D class.

# Import detail

```
from ansys.geometry.core.math.point import Point2D
```

# **Property detail**

```
property Point2D.x: pint.Quantity
```

X plane component value.

property Point2D.y: pint.Quantity

Y plane component value.

### **Attribute detail**

Point2D.flat

#### **Method detail**

```
Point2D.__eq__(other: Point2D) \rightarrow bool
```

Equals operator for the Point2D class.

Point2D.\_\_ne\_\_(other: Point2D)  $\rightarrow$  bool

Not equals operator for the Point2D class.

Point2D. **\_\_add\_\_**( $other: Point2D \mid ansys.geometry.core.math.vector.Vector2D) <math>\rightarrow Point2D$ 

Add operation for the Point2D class.

 $\texttt{Point2D}.\_\_\texttt{sub}\_\_(\textit{other:} \ \texttt{Point2D}) \rightarrow \textit{Point2D}$ 

Subtraction operation for the Point2D class.

Point2D.unit()  $\rightarrow$  pint.Unit

Get the unit of the object.

Point2D.base\_unit()  $\rightarrow$  pint.Unit

Get the base unit of the object.

### Point3D

```
class ansys.geometry.core.math.point.Point3D(input: numpy.ndarray |
```

ansys.geometry.core.typing.RealSequence = DEFAULT\_POINT3D\_VALUES, unit: pint.Unit | None = None)

Bases: numpy.ndarray, ansys.geometry.core.misc.units.PhysicalQuantity

Provides geometry primitive representation for a 3D point.

#### **Parameters**

#### input

[ndarray | RealSequence, default: DEFAULT\_POINT3D\_VALUES] Direction arguments, either as a numpy.ndarray class or as a RealSequence.

#### unit

[Unit | None, default: DEFAULT\_UNITS.LENGTH] Units for defining 3D point values. If not specified, the default unit is DEFAULT\_UNITS.LENGTH.

# **Overview**

### **Methods**

unit	Get the unit of the object.
base_unit	Get the base unit of the object.
transform	Transform the 3D point with a transformation matrix.

# **Properties**

- x X plane component value.
- y Y plane component value.
- z Z plane component value.

### **Attributes**

flat

# **Special methods**

eq	Equals operator for the Point3D class.
ne	Not equals operator for the Point3D class.
add	Add operation for the Point3D class.
sub	Subtraction operation for the Point3D class.

# Import detail

```
from ansys.geometry.core.math.point import Point3D
```

# **Property detail**

```
property Point3D.x: pint.Quantity
    X plane component value.
property Point3D.y: pint.Quantity
    Y plane component value.
property Point3D.z: pint.Quantity
```

Z plane component value.

# **Attribute detail**

Point3D.flat

### **Method detail**

Point3D.\_\_eq\_\_(other: Point3D) → bool Equals operator for the Point3D class. Point3D.\_\_ne\_\_(other: Point3D) → bool

Not equals operator for the Point3D class.

Point3D.\_\_add\_\_(other: Point3D | ansys.geometry.core.math.vector.Vector3D)  $\rightarrow$  Point3D Add operation for the Point3D class.

Point3D.\_\_sub\_\_(other: Point3D | ansys.geometry.core.math.vector.Vector3D)  $\rightarrow$  Point3D Subtraction operation for the Point3D class.

Point3D.unit()  $\rightarrow$  pint.Unit

Get the unit of the object.

Point3D.base\_unit()  $\rightarrow$  pint.Unit

Get the base unit of the object.

Point3D.**transform**(*matrix*: ansys.geometry.core.math.matrix.Matrix44) → *Point3D* 

Transform the 3D point with a transformation matrix.

#### **Parameters**

#### matrix

[Matrix44] 4x4 transformation matrix to apply to the point.

#### Returns

#### Point3D

New 3D point that is the transformed copy of the original 3D point after applying the transformation matrix.

#### **Notes**

Transform the Point3D object by applying the specified 4x4 transformation matrix and return a new Point3D object representing the transformed point.

# **Description**

Provides geometry primitive representation for 2D and 3D points.

# Module detail

### point.DEFAULT\_POINT2D\_VALUES

Default values for a 2D point.

# point.DEFAULT\_POINT3D\_VALUES

Default values for a 3D point.

# point.BASE\_UNIT\_LENGTH

Default value for the length of the base unit.

# The vector.py module

# **Summary**

### **Classes**

Vector3D	Provides for managing and creating a 3D vector.
Vector2D	Provides for creating and managing a 2D vector.
UnitVector3D	Provides for creating and managing a 3D unit vector.
UnitVector2D	Provides for creating and managing a 3D unit vector.

# Vector3D

Bases: numpy.ndarray

Provides for managing and creating a 3D vector.

# **Parameters**

### input

[ndarray | RealSequence] 3D numpy.ndarray class with shape(X,).

### **Overview**

### **Constructors**

# **Methods**

is_perpendicular_to	Check if this vector and another vector are perpendicular.
is_parallel_to	Check if this vector and another vector are parallel.
is_opposite	Check if this vector and another vector are opposite.
normalize	Return a normalized version of the 3D vector.
transform	Transform the 3D vector3D with a transformation matrix.
get_angle_between	Get the angle between this 3D vector and another 3D vector.
cross	Return the cross product of Vector3D objects.

# **Properties**

X	X coordinate of the Vector3D class.
у	Y coordinate of the Vector3D class.
Z	Z coordinate of the Vector3D class.
norm	Norm of the vector.
magnitude	Norm of the vector.
is_zero	Check if all components of the 3D vector are zero.

### Special methods

eq	Equals operator for the Vector3D class.
ne	Not equals operator for the Vector3D class.
mu1	Overload * operator with dot product.
mod	Overload % operator with cross product.
add	Addition operation overload for 3D vectors.
sub	Subtraction operation overload for 3D vectors.

# Import detail

```
from ansys.geometry.core.math.vector import Vector3D
```

# **Property detail**

```
property Vector3D.x: ansys.geometry.core.typing.Real
     X coordinate of the Vector3D class.
property Vector3D.y: ansys.geometry.core.typing.Real
     Y coordinate of the Vector3D class.
property Vector3D.z: ansys.geometry.core.typing.Real
     Z coordinate of the Vector3D class.
property Vector3D.norm: float
```

Norm of the vector.

property Vector3D.magnitude: float

Norm of the vector.

property Vector3D.is\_zero: bool

Check if all components of the 3D vector are zero.

#### **Method detail**

```
Vector3D.is_perpendicular_to(other\_vector: Vector3D) \rightarrow bool
```

Check if this vector and another vector are perpendicular.

Vector3D.is\_parallel\_to( $other\_vector$ : Vector3D)  $\rightarrow$  bool

Check if this vector and another vector are parallel.

 $Vector3D.is\_opposite(other\_vector: Vector3D) \rightarrow bool$ 

Check if this vector and another vector are opposite.

Vector3D.**normalize**()  $\rightarrow$  *Vector3D* 

Return a normalized version of the 3D vector.

Vector3D.transform(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Vector3D$ 

Transform the 3D vector3D with a transformation matrix.

#### **Parameters**

### matrix

[Matrix44] 4x4 transformation matrix to apply to the vector.

Returns

#### Vector3D

A new 3D vector that is the transformed copy of the original 3D vector after applying the transformation matrix.

#### **Notes**

Transform the Vector3D object by applying the specified 4x4 transformation matrix and return a new Vector3D object representing the transformed vector.

```
Vector3D.get_angle_between(v: Vector3D) \rightarrow pint.Quantity
```

Get the angle between this 3D vector and another 3D vector.

#### **Parameters**

V

[Vector3D] Other 3D vector for computing the angle.

#### Returns

# Quantity

Angle between these two 3D vectors.

Vector3D.cross(v: Vector3D)  $\rightarrow Vector3D$ 

Return the cross product of Vector3D objects.

Vector3D.**\_\_eq**\_\_(other: Vector3D)  $\rightarrow$  bool

Equals operator for the Vector3D class.

Vector3D.\_\_ne\_\_(other: Vector3D)  $\rightarrow$  bool

Not equals operator for the Vector3D class.

 $\label{lem:vector3D} \begin{tabular}{ll} Vector3D & ansys.geometry.core.typing.Real \end{tabular} \rightarrow Vector3D & ansys.geometry.core.typing.Real \end{tabular}$ 

Overload \* operator with dot product.

### **Notes**

This method also admits scalar multiplication.

```
Vector3D.__mod__(other: Vector3D) \rightarrow Vector3D
```

Overload % operator with cross product.

 $\label{lem:vector3D} \begin{tabular}{ll} Vector3D & ansys.geometry.core.math.point.Point3D \end{tabular} \rightarrow \textit{Vector3D} & ansys.geometry.core.math.point.Point3D \end{tabular}$ 

Addition operation overload for 3D vectors.

Vector3D.**\_\_sub\_\_**( $other: Vector3D) \rightarrow Vector3D$ 

Subtraction operation overload for 3D vectors.

Create a 3D vector from two distinct 3D points.

#### **Parameters**

#### point\_a

[ndarray | RealSequence | Point 3D] *Point 3D* class representing the first point.

# point\_b

[ndarray | RealSequence | Point3D] *Point3D* class representing the second point.

#### Returns

### Vector3D

3D vector from point\_a to point\_b.

### **Notes**

The resulting 3D vector is always expressed in Point3D base units.

### Vector2D

Bases: numpy.ndarray

Provides for creating and managing a 2D vector.

### **Parameters**

### input

 $[ndarray \mid \texttt{RealSequence}] \ 2D \ numpy . \ ndarray \ class \ with \ shape(X,).$ 

#### Overview

### **Constructors**

from_points Create a 2D vector from two distinct 2	2D points.
----------------------------------------------------	------------

#### **Methods**

cross	Return the cross product of Vector2D objects.
is_perpendicular_to	Check if this 2D vector and another 2D vector are perpendicular.
is_parallel_to	Check if this vector and another vector are parallel.
is_opposite	Check if this vector and another vector are opposite.
normalize	Return a normalized version of the 2D vector.
get_angle_between	Get the angle between this 2D vector and another 2D vector.

# **Properties**

X	X coordinate of the 2D vector.
у	Y coordinate of the 2D vector.
norm	Norm of the 2D vector.
magnitude	Norm of the 2D vector.
is_zero	Check if values for all components of the 2D vector are zero.

# **Special methods**

eq	Equals operator for the Vector2D class.
ne	Not equals operator for the Vector2D class.
mu1	Overload * operator with dot product.
add	Addition operation overload for 2D vectors.
sub	Subtraction operation overload for 2D vectors.
mod	Overload % operator with cross product.

# Import detail

```
from ansys.geometry.core.math.vector import Vector2D
```

# **Property detail**

```
property Vector2D.x: ansys.geometry.core.typing.Real
    X coordinate of the 2D vector.
```

```
property Vector2D.y: ansys.geometry.core.typing.Real
```

Y coordinate of the 2D vector.

property Vector2D.norm: float

Norm of the 2D vector.

property Vector2D.magnitude: float

Norm of the 2D vector.

property Vector2D.is\_zero: bool

Check if values for all components of the 2D vector are zero.

### **Method detail**

 $\mbox{Vector2D.} \ \mbox{\bf cross}(\nu : \mbox{Vector2D}) \rightarrow \mbox{ansys.geometry.} \mbox{core.} \mbox{typing.} \mbox{Real}$  Return the cross product of Vector2D objects.

Vector2D.is\_perpendicular\_to(other\_vector: Vector2D) → bool Check if this 2D vector and another 2D vector are perpendicular.

Vector2D.is\_parallel\_to(other\_vector: Vector2D)  $\rightarrow$  bool Check if this vector and another vector are parallel.

Vector2D.is\_opposite( $other\_vector$ : Vector2D)  $\rightarrow$  bool Check if this vector and another vector are opposite.

 $Vector2D.normalize() \rightarrow Vector2D$ 

Return a normalized version of the 2D vector.

Vector2D.**get\_angle\_between**(v: Vector2D)  $\rightarrow$  pint.Quantity Get the angle between this 2D vector and another 2D vector.

#### **Parameters**

V

[Vector2D] Other 2D vector to compute the angle with.

# Returns

```
Quantity
                    Angle between these two 2D vectors.
Vector2D.__eq__(other: Vector2D) \rightarrow bool
      Equals operator for the Vector2D class.
Vector2D.__ne__(other: Vector2D) \rightarrow bool
      Not equals operator for the Vector2D class.
Vector2D.\__mul\_\_(other: Vector2D \mid ansys.geometry.core.typing.Real) \rightarrow Vector2D \mid
                     ansys.geometry.core.typing.Real
      Overload * operator with dot product.
      Notes
      This method also admits scalar multiplication.
Vector2D. __add__(other: Vector2D | ansys.geometry.core.math.point.Point2D) \rightarrow Vector2D |
                     ansys.geometry.core.math.point.Point2D
      Addition operation overload for 2D vectors.
Vector2D.__sub__(other: Vector2D) \rightarrow Vector2D
      Subtraction operation overload for 2D vectors.
Vector2D.\_\_mod\_\_(other: Vector2D) \rightarrow ansys.geometry.core.typing.Real
      Overload % operator with cross product.
classmethod Vector2D. from_points(point_a: numpy.ndarray | ansys.geometry.core.typing.RealSequence |
                                         ansys.geometry.core.math.point.Point2D, point_b: numpy.ndarray
                                         ansys.geometry.core.typing.RealSequence |
                                          ansys.geometry.core.math.point.Point2D)
      Create a 2D vector from two distinct 2D points.
           Parameters
               point a
                    [ndarray | RealSequence | Point2D] Point2D class representing the first point.
               point b
                    [ndarray | RealSequence | Point2D] Point2D class representing the second point.
           Returns
               Vector2D
                    2D vector from point_a to point_b.
      Notes
      The resulting 2D vector is always expressed in Point2D base units.
UnitVector3D
class ansys.geometry.core.math.vector.UnitVector3D(shape, dtype=float, buffer=None, offset=0,
                                                                strides=None, order=None)
      Bases: Vector3D
      Provides for creating and managing a 3D unit vector.
```

**Parameters** 

# input

[ndarray | RealSequence | Vector3D]

- 1D numpy.ndarray class with shape(X,)
- Vector3D

#### Overview

#### **Constructors**

# Import detail

```
from ansys.geometry.core.math.vector import UnitVector3D
```

### **Method detail**

Create a 3D unit vector from two distinct 3D points.

```
Parameters
```

```
point_a
```

[ndarray | RealSequence | Point3D] *Point3D* class representing the first point.

point b

[ndarray | RealSequence | Point3D] Point3D class representing the second point.

### Returns

### UnitVector3D

3D unit vector from point\_a to point\_b.

#### UnitVector2D

Bases: Vector2D

Provides for creating and managing a 3D unit vector.

#### **Parameters**

#### input

[ndarray | RealSequence | Vector2D]

- 1D numpy.ndarray class with shape(X,)
- Vector2D

# **Overview**

#### **Constructors**

# Import detail

```
from ansys.geometry.core.math.vector import UnitVector2D
```

# **Method detail**

Create a 2D unit vector from two distinct 2D points.

#### **Parameters**

point\_a

[ndarray | RealSequence | Point2D] *Point2D* class representing the first point.

point\_b

[ndarray | RealSequence | Point2D] Point2D class representing the second point.

### Returns

# UnitVector2D

2D unit vector from point\_a to point\_b.

# **Description**

Provides for creating and managing 2D and 3D vectors.

# **Description**

PyAnsys Geometry math subpackage.

# The misc package

### **Summary**

### **Submodules**

accuracy	Provides for evaluating decimal precision.	
auxiliary	Auxiliary functions for the PyAnsys Geometry library.	
checks	Provides functions for performing common checks.	
measurements	Provides various measurement-related classes.	
options	Provides various option classes.	
units	Provides for handling units homogeneously throughout PyAnsys Geometry.	

# The accuracy.py module

# **Summary**

### **Classes**

Accuracy	Decimal precision evaluations for math operations.
----------	----------------------------------------------------

# **Constants**

LENGTH_ACCURACY	Constant for decimal accuracy in length comparisons.
ANGLE_ACCURACY	Constant for decimal accuracy in angle comparisons.
DOUBLE_ACCURACY	Constant for double accuracy.

# Accuracy

**class** ansys.geometry.core.misc.accuracy.**Accuracy**Decimal precision evaluations for math operations.

### Overview

#### Static methods

length_accuracy	Return the LENGTH_ACCURACY constant.
angle_accuracy	Return the ANGLE_ACCURACY constant.
double_accuracy	Return the DOUBLE_ACCURACY constant.
length_is_equal	Check if the comparison length is equal to the reference length.
equal_doubles	Compare two double values.
compare_with_tolerance	Compare two doubles given the relative and absolute tolerances.
<pre>length_is_greater_than_or_equal</pre>	Check if the length is greater than the reference length.
<pre>length_is_less_than_or_equal</pre>	Check if the length is less than or equal to the reference length.
length_is_zero	Check if the length is within the length accuracy of exact zero.
length_is_negative	Check if the length is below a negative length accuracy.
length_is_positive	Check if the length is above a positive length accuracy.
angle_is_zero	Check if the length is within the angle accuracy of exact zero.
angle_is_negative	Check if the angle is below a negative angle accuracy.
angle_is_positive	Check if the angle is above a positive angle accuracy.
is_within_tolerance	Check if two values are inside a relative and absolute tolerance.

# Import detail

from ansys.geometry.core.misc.accuracy import Accuracy

# **Method detail**

 $\mbox{\bf static Accuracy.length\_accuracy()} \rightarrow \mbox{ansys.geometry.core.typing.Real} \\ \mbox{Return the LENGTH\_ACCURACY constant.} \\$ 

 $\mbox{\bf static Accuracy.angle\_accuracy()} \rightarrow \mbox{ansys.geometry.core.typing.Real} \\ \mbox{Return the ANGLE\_ACCURACY constant.} \\$ 

 $\textbf{static} \ \, \textbf{Accuracy}. \textbf{double\_accuracy}() \rightarrow ansys. geometry. core. typing. Real$ 

Return the DOUBLE\_ACCURACY constant.

**static** Accuracy.length\_is\_equal(comparison\_length: ansys.geometry.core.typing.Real, reference\_length: ansys.geometry.core.typing.Real)  $\rightarrow$  bool

Check if the comparison length is equal to the reference length.

#### Returns

bool

True if the comparison length is equal to the reference length within the length accuracy, False otherwise.

#### **Notes**

The check is done up to the constant value specified for LENGTH\_ACCURACY.

**static** Accuracy.equal\_doubles(a: ansys.geometry.core.typing.Real, b: ansys.geometry.core.typing.Real)

Compare two double values.

 $\begin{tabular}{lll} \textbf{static} & Accuracy. \textbf{compare\_with\_tolerance}(a: ansys. geometry. core. typing. Real, b: \\ & ansys. geometry. core. typing. Real, relative\_tolerance: \\ & ansys. geometry. core. typing. Real, absolute\_tolerance: \\ & ansys. geometry. core. typing. Real) \rightarrow \end{tabular}$ 

Compare two doubles given the relative and absolute tolerances.

static Accuracy.length\_is\_greater\_than\_or\_equal(comparison\_length: ansys.geometry.core.typing.Real, reference\_length: ansys.geometry.core.typing.Real)  $\rightarrow$  bool

Check if the length is greater than the reference length.

#### Returns

bool

True if the comparison length is greater than the reference length within the length accuracy, False otherwise.

ansys.geometry.core.typing.Real

### **Notes**

The check is done up to the constant value specified for LENGTH\_ACCURACY.

Check if the length is less than or equal to the reference length.

# Returns

bool

True if the comparison length is less than or equal to the reference length within the length accuracy, False otherwise.

#### **Notes**

The check is done up to the constant value specified for LENGTH\_ACCURACY.

**static** Accuracy.**length\_is\_zero**( $length: ansys.geometry.core.typing.Real) <math>\rightarrow$  bool Check if the length is within the length accuracy of exact zero.

#### Returns

bool

True if the length is within the length accuracy of exact zero, False otherwise.

static Accuracy.length\_is\_negative(length: ansys.geometry.core.typing.Real)  $\rightarrow$  bool Check if the length is below a negative length accuracy.

#### Returns

bool

**True** if the length is below a negative length accuracy, False otherwise.

**static** Accuracy.**length\_is\_positive**( $length: ansys.geometry.core.typing.Real) <math>\rightarrow$  bool Check if the length is above a positive length accuracy.

#### Returns

bool

True if the length is above a positive length accuracy, False otherwise.

**static** Accuracy.**angle\_is\_zero**( $angle: ansys.geometry.core.typing.Real) <math>\rightarrow$  bool Check if the length is within the angle accuracy of exact zero.

#### Returns

bool

True if the length is within the angle accuracy of exact zero, False otherwise.

static Accuracy.angle\_is\_negative(angle: ansys.geometry.core.typing.Real)  $\rightarrow$  bool Check if the angle is below a negative angle accuracy.

### Returns

bool

True if the angle is below a negative angle accuracy,

False otherwise.

static Accuracy.angle\_is\_positive(angle: ansys.geometry.core.typing.Real)  $\rightarrow$  bool Check if the angle is above a positive angle accuracy.

#### Returns

bool

True if the angle is above a positive angle accuracy,

False otherwise.

static Accuracy.is\_within\_tolerance(a: ansys.geometry.core.typing.Real, b: ansys.geometry.core.typing.Real, relative\_tolerance: ansys.geometry.core.typing.Real, absolute\_tolerance: ansys.geometry.core.typing.Real)  $\rightarrow$  bool

Check if two values are inside a relative and absolute tolerance.

#### **Parameters**

a

[Real] First value.

b

[Real] Second value.

# relative\_tolerance

[Real] Relative tolerance accepted.

# $absolute\_tolerance$

[Real] Absolute tolerance accepted.

#### **Returns**

bool

True if the values are inside the accepted tolerances, False otherwise.

# **Description**

Provides for evaluating decimal precision.

### Module detail

accuracy.LENGTH\_ACCURACY: ansys.geometry.core.typing.Real = 1e-08

Constant for decimal accuracy in length comparisons.

accuracy.ANGLE\_ACCURACY: ansys.geometry.core.typing.Real = 1e-06

Constant for decimal accuracy in angle comparisons.

accuracy.DOUBLE\_ACCURACY: ansys.geometry.core.typing.Real = 1e-13

Constant for double accuracy.

# The auxiliary.py module

### **Summary**

### **Functions**

<pre>get_design_from_component</pre>	Get the Design of the given Component object.
<pre>get_design_from_body</pre>	Get the Design of the given Body object.
<pre>get_design_from_face</pre>	Get the Design of the given Face object.
<pre>get_design_from_edge</pre>	Get the Design of the given Edge object.
<pre>get_all_bodies_from_design</pre>	Find all the Body objects inside a Design.
<pre>get_bodies_from_ids</pre>	Find the Body objects inside a Design from its ids.
<pre>get_faces_from_ids</pre>	Find the Face objects inside a Design from its ids.
<pre>get_edges_from_ids</pre>	Find the Edge objects inside a Design from its ids.
convert_color_to_hex	Get the hex string color from input formats.

## **Constants**

DEFAULT\_COLOR

# **Description**

Auxiliary functions for the PyAnsys Geometry library.

```
Module detail
```

```
\textbf{auxiliary.get\_design\_from\_component}(component: ansys.geometry.core.designer.component.Component)} \\ \rightarrow ansys.geometry.core.designer.design.Design
```

Get the Design of the given Component object.

#### **Parameters**

### component

[Component] The component object for which to find the Design.

### Returns

Design

The Design of the provided component object.

 $\textbf{auxiliary.get\_design\_from\_body}(\textit{body}: ansys.geometry.core.designer.body.Body}) \rightarrow \textit{ansys.geometry.core.designer.design.Design}$ 

Get the Design of the given Body object.

#### **Parameters**

body

[Body] The body object for which to find the Design.

#### Returns

Design

The Design of the provided body object.

 $auxiliary. \textbf{get\_design\_from\_face} (\textit{face:} \ ansys. \textit{geometry.core.designer.face}. Face) \rightarrow \textit{ansys.geometry.core.designer.design.Design}$ 

Get the Design of the given Face object.

#### **Parameters**

face

[Face] The face object for which to find the Design.

### Returns

Design

The Design of the provided face object.

 $\hbox{auxiliary.} \textbf{get\_design\_from\_edge}(\textit{edge}: \textit{ansys.} \textit{geometry.} \textit{core.} \textit{designer.} \textit{edge}. \textit{Edge}) \rightarrow \textit{ansys.} \textit{geometry.} \textit{core.} \textit{designer.} \textit{design}. \textit{Design}$ 

Get the Design of the given Edge object.

### **Parameters**

edge

[Edge] The edge object for which to find the Design.

### Returns

Design

The Design of the provided edge object.

```
auxiliary.get_all_bodies_from_design(design: ansys.geometry.core.designer.design.Design) →
                                              list[ansys.geometry.core.designer.body.Body]
     Find all the Body objects inside a Design.
           Parameters
               design
                   [Design] Parent design for the bodies.
           Returns
               list[Body]
                   List of Body objects.
     Notes
     This method takes a design and gets the corresponding Body objects.
auxiliary.get_bodies_from_ids(design: ansys.geometry.core.designer.design.Design, body_ids: list[str]) →
                                     list[ansys.geometry.core.designer.body.Body]
     Find the Body objects inside a Design from its ids.
           Parameters
               design
                   [Design] Parent design for the faces.
               body_ids
                   [list[str]] List of body ids.
           Returns
               list[Body]
                   List of Body objects.
     Notes
     This method takes a design and body ids, and gets their corresponding Body object.
auxiliary.get_faces_from_ids(design: ansys.geometry.core.designer.design.Design, face_ids: list[str]) →
                                    list[ansys.geometry.core.designer.face.Face]
     Find the Face objects inside a Design from its ids.
           Parameters
               design
                   [Design] Parent design for the faces.
               face ids
                   [list[str]] List of face ids.
           Returns
               list[Face]
                   List of Face objects.
```

# Notes

This method takes a design and face ids, and gets their corresponding Face object.

```
auxiliary.get_edges_from_ids(design: ansys.geometry.core.designer.design.Design, edge_ids: list[str]) \rightarrow
                                   list[ansys.geometry.core.designer.edge.Edge]
     Find the Edge objects inside a Design from its ids.
           Parameters
               design
                   [Design] Parent design for the edges.
               edge_ids
                   [list[str]] List of edge ids.
           Returns
               list[Edge]
                   List of Edge objects.
     Notes
     This method takes a design and edge ids, and gets their corresponding Edge objects.
auxiliary.convert_color_to_hex(color: str | tuple[float, float, float]) → str
     Get the hex string color from input formats.
           Parameters
               color
                   [str | tuple[float, float]] Color to set the body to. This can be a string repre-
                   senting a color name or a tuple of RGB values in the range [0, 1] (RGBA) or [0, 255] (pure
                   RGB).
           Returns
                   The hex code string for the color.
auxiliary.DEFAULT_COLOR = '#D6F7D1'
The checks.py module
```

**Summary** 

# **Functions**

ensure_design_is_active	Make sure that the design is active before executing a method.
<pre>check_is_float_int</pre>	Check if a parameter has a float or integer value.
<pre>check_ndarray_is_float_int</pre>	Check if a numpy.ndarray has float/integer types.
<pre>check_ndarray_is_not_none</pre>	Check if a numpy.ndarray is all None.
<pre>check_ndarray_is_all_nan</pre>	Check if a numpy.ndarray is all nan-valued.
<pre>check_ndarray_is_non_zero</pre>	Check if a numpy.ndarray is zero-valued.
<pre>check_pint_unit_compatibility</pre>	Check if input pint. Unit is compatible with the expected input.
<pre>check_type_equivalence</pre>	Check if an input object is of the same class as an expected object.
check_type	Check if an input object is of the same type as expected types.
<pre>check_type_all_elements_in_iterable</pre>	Check if all elements in an iterable are of the same type as expected.
min_backend_version	Compare a minimum required version to the current backend ver-
	sion.
deprecated_method	Decorate a method as deprecated.
deprecated_argument	Decorate a method argument as deprecated.
<pre>run_if_graphics_required</pre>	Check if graphics are available.
graphics_required	Decorate a method as requiring graphics.

### **Constants**

ERROR\_GRAPHICS\_REQUIRED Message to display when graphics are required for a method.

### **Description**

Provides functions for performing common checks.

### Module detail

### checks.ensure\_design\_is\_active(method)

Make sure that the design is active before executing a method.

This function is necessary to be called whenever we do any operation on the design. If we are just accessing information of the class, it is not necessary to call this.

checks.check\_is\_float\_int( $param: object, param\_name: str \mid None = None) \rightarrow None$ 

Check if a parameter has a float or integer value.

#### **Parameters**

#### param

[object] Object instance to check.

### param\_name

[str, default: None] Parameter name (if any).

#### Raises

### **TypeError**

If the parameter does not have a float or integer value.

checks.check\_ndarray\_is\_float\_int(param: numpy.ndarray, param\_name: str | None = None) → None Check if a numpy.ndarray has float/integer types.

#### **Parameters**

```
param
                   [ndarray] numpy.ndarray instance to check.
               param name
                   [str, default: None] numpy.ndarray instance name (if any).
           Raises
               TypeError
                   If the numpy . ndarray instance does not have float or integer values.
checks.check_ndarray_is_not_none(param: numpy.ndarray, param_name: str | None = None) <math>\rightarrow None
     Check if a numpy.ndarray is all None.
           Parameters
               param
                   [ndarray] numpy.ndarray instance to check.
               param name
                   [str, default: None] numpy.ndarray instance name (if any).
           Raises
               ValueError
                   If the numpy .ndarray instance has a value of None for all parameters.
checks.check_ndarray_is_all_nan(param: numpy.ndarray, param_name: str \mid None = None) <math>\rightarrow None
     Check if a numpy.ndarray is all nan-valued.
           Parameters
               param
                   [ndarray] numpy.ndarray instance to check.
               param name
                   [str or None, default: None] numpy.ndarray instance name (if any).
           Raises
               ValueError
                   If the numpy.ndarray instance is all nan-valued.
checks.check_ndarray_is_non_zero(param: numpy.ndarray, param_name: str | None = None) <math>\rightarrow None
     Check if a numpy.ndarray is zero-valued.
           Parameters
               param
                   [ndarray] numpy.ndarray instance to check.
               param name
                   [str, default: None] numpy.ndarray instance name (if any).
           Raises
               ValueError
                   If the numpy.ndarray instance is zero-valued.
checks.check_pint_unit_compatibility(input: pint.Unit, expected: pint.Unit) \rightarrow None
     Check if input pint. Unit is compatible with the expected input.
           Parameters
```

```
input
                    [Unit] pint. Unit input.
                expected
                    [Unit] pint.Unit expected dimensionality.
           Raises
                TypeError
                    If the input is not compatible with the pint. Unit class.
checks.check_type_equivalence(input: object, expected: object) \rightarrow None
      Check if an input object is of the same class as an expected object.
           Parameters
                input
                    [object] Input object.
                expected
                    [object] Expected object.
           Raises
                TypeError
                    If the objects are not of the same class.
checks.check_type(input: object, expected_type: type | tuple[type, Ellipsis]) \rightarrow None
      Check if an input object is of the same type as expected types.
           Parameters
                input
                    [object] Input object.
                expected_type
                    [type | tuple[type, ...]] One or more types to compare the input object against.
           Raises
                TypeError
                    If the object does not match the one or more expected types.
checks.check_type_all_elements_in_iterable(input: collections.abc.Iterable, expected_type: type |
                                                       tuple[type, Ellipsis]) \rightarrow None
      Check if all elements in an iterable are of the same type as expected.
           Parameters
                input
                    [Iterable] Input iterable.
                expected_type
                    [type | tuple[type, ...]] One or more types to compare the input object against.
           Raises
                TypeError
                    If one of the elements in the iterable does not match the one or more expected types.
checks.min_backend_version(major: int, minor: int, service_pack: int)
      Compare a minimum required version to the current backend version.
```

**Parameters** 

#### major

[int] Minimum major version required by the method.

#### minor

[int] Minimum minor version required by the method.

### service\_pack

[int] Minimum service pack version required by the method.

#### Raises

# ${\tt GeometryRuntimeError}$

If the method version is higher than the backend version.

### GeometryRuntimeError

If the client is not available.

```
checks.deprecated_method(alternative: str \mid None = None, info: str \mid None = None, version: str \mid None = None, version: str \mid None = None)
```

Decorate a method as deprecated.

#### **Parameters**

#### alternative

[str, default: None] Alternative method to use. If provided, the warning message will include the alternative method.

#### info

[str, default: None] Additional information to include in the warning message.

#### version

[str, default: None] Version where the method was deprecated.

#### remove

[str, default: None] Version where the method will be removed.

```
checks.deprecated_argument(arg: str, alternative: str \mid None = None, info: str \mid None = None, version: str \mid None = None, version: str \mid None = None)
```

Decorate a method argument as deprecated.

#### **Parameters**

# arg

[str] Argument to mark as deprecated.

### alternative

[str, default: None] Alternative argument to use. If provided, the warning message will include the alternative argument.

#### info

[str, default: None] Additional information to include in the warning message.

#### version

[str, default: None] Version where the method was deprecated.

### remove

[str, default: None] Version where the method will be removed.

# checks.run\_if\_graphics\_required()

Check if graphics are available.

### checks.graphics\_required(method)

Decorate a method as requiring graphics.

#### **Parameters**

#### method

[callable()] Method to decorate.

### **Returns**

callable()

Decorated method.

checks.ERROR\_GRAPHICS\_REQUIRED = 'Graphics are required for this method. Please install
the ``graphics`` target to use this...

Message to display when graphics are required for a method.

### The measurements.py module

### **Summary**

### **Classes**

SingletonMeta	Provides a thread-safe implementation of a singleton design pattern.
DefaultUnitsClass	Provides default units for PyAnsys Geometry.
Measurement	Provides the PhysicalQuantity subclass for holding a measurement.
Distance	Provides the Measurement subclass for holding a distance.
Angle	Provides the Measurement subclass for holding an angle.

### **Constants**

DEFAULT\_UNITS PyAnsys Geometry default units object.

# SingletonMeta

class ansys.geometry.core.misc.measurements.SingletonMeta

Bases: type

Provides a thread-safe implementation of a singleton design pattern.

#### Overview

# **Special methods**

\_\_call\_\_ Return a single instance of the class.

### Import detail

from ansys.geometry.core.misc.measurements import SingletonMeta

# **Method detail**

```
SingletonMeta.__call__(*args, **kwargs)
```

Return a single instance of the class.

Possible changes to the value of the \_\_init\_\_ argument do not affect the returned instance.

### DefaultUnitsClass

class ansys.geometry.core.misc.measurements.DefaultUnitsClass
 Provides default units for PyAnsys Geometry.

#### Overview

### **Properties**

LENGTH	Default length unit for PyAnsys Geometry.
ANGLE	Default angle unit for PyAnsys Geometry.
SERVER_LENGTH	Default length unit for gRPC messages.
SERVER_AREA	Default area unit for gRPC messages.
SERVER_VOLUME	Default volume unit for gRPC messages.
SERVER_ANGLE	Default angle unit for gRPC messages.

# Import detail

from ansys.geometry.core.misc.measurements import DefaultUnitsClass

# **Property detail**

```
property DefaultUnitsClass.LENGTH: pint.Unit
```

Default length unit for PyAnsys Geometry.

property DefaultUnitsClass.ANGLE: pint.Unit

Default angle unit for PyAnsys Geometry.

property DefaultUnitsClass.SERVER\_LENGTH: pint.Unit

Default length unit for gRPC messages.

#### **Notes**

The default units on the server side are not modifiable yet.

property DefaultUnitsClass.SERVER\_AREA: pint.Unit

Default area unit for gRPC messages.

#### **Notes**

The default units on the server side are not modifiable yet.

property DefaultUnitsClass.SERVER\_VOLUME: pint.Unit

Default volume unit for gRPC messages.

### **Notes**

The default units on the server side are not modifiable yet.

```
property DefaultUnitsClass.SERVER_ANGLE: pint.Unit
```

Default angle unit for gRPC messages.

### **Notes**

The default units on the server side are not modifiable yet.

#### Measurement

Bases: ansys.geometry.core.misc.units.PhysicalQuantity

Provides the PhysicalQuantity subclass for holding a measurement.

### **Parameters**

#### value

[Real | Quantity] Value of the measurement.

#### unit

[Unit] Units for the measurement.

#### dimensions

[Unit] Units for extracting the dimensions of the measurement. If ~pint.Unit.meter is given, the dimension extracted is [length].

### **Overview**

# **Properties**

value Value of the measurement.

# **Special methods**

repr	Representation of the Measurement class.
eq	Equals operator for the Measurement class.

# Import detail

from ansys.geometry.core.misc.measurements import Measurement

### **Property detail**

property Measurement.value: pint.Quantity

Value of the measurement.

# **Method detail**

```
Measurement.__repr__()
Representation of the Measurement class.

Measurement.__eq__(other: Measurement) \rightarrow bool
Equals operator for the Measurement class.
```

#### **Distance**

```
class ansys.geometry.core.misc.measurements.Distance(value: ansys.geometry.core.typing.Real | pint.Quantity, unit: pint.Unit | None = None)
```

Bases: Measurement

Provides the Measurement subclass for holding a distance.

#### **Parameters**

```
value
   [Real | Quantity] Value of the distance.
unit
   [Unit, default: DEFAULT_UNITS.LENGTH] Units for the distance.
```

# Import detail

```
from ansys.geometry.core.misc.measurements import Distance
```

#### **Angle**

```
class ansys.geometry.core.misc.measurements.Angle(value: ansys.geometry.core.typing.Real \mid pint.Quantity, unit: pint.Unit \mid None = None)
```

Bases: Measurement

Provides the Measurement subclass for holding an angle.

### **Parameters**

```
value
    [Real | Quantity] Value of the angle.
unit
    [Unit, default: DEFAULT_UNITS.ANGLE] Units for the distance.
```

# Import detail

```
from ansys.geometry.core.misc.measurements import Angle
```

# **Description**

Provides various measurement-related classes.

### Module detail

#### measurements.DEFAULT\_UNITS

PyAnsys Geometry default units object.

# The options.py module

### **Summary**

#### **Classes**

*ImportOptions* Import options when opening a file.

# ImportOptions

class ansys.geometry.core.misc.options.ImportOptions

Import options when opening a file.

### **Parameters**

# cleanup\_bodies

[bool = False] Simplify geometry and clean up topology.

# import\_coordinate\_systems

[bool = False] Import coordinate systems.

### import\_curves

[bool = False] Import curves.

# $import\_hidden\_components\_and\_geometry$

[bool = False] Import hidden components and geometry.

### import\_names

[bool = False] Import names of bodies and curves.

### import\_planes

[bool = False] Import planes.

# import\_points

[bool = False] Import points.

### **Overview**

# **Methods**

*to\_dict* Provide the dictionary representation of the ImportOptions class.

### **Attributes**

```
cleanup_bodies
import_coordinate_systems
import_curves
import_hidden_components_and_geometry
import_names
import_planes
import_points
```

# Import detail

```
from ansys.geometry.core.misc.options import ImportOptions
```

### **Attribute detail**

```
ImportOptions.cleanup_bodies: bool = False
ImportOptions.import_coordinate_systems: bool = False
ImportOptions.import_curves: bool = False
ImportOptions.import_hidden_components_and_geometry: bool = False
ImportOptions.import_names: bool = False
ImportOptions.import_planes: bool = False
ImportOptions.import_points: bool = False
```

### Method detail

ImportOptions.to\_dict()

Provide the dictionary representation of the ImportOptions class.

# **Description**

Provides various option classes.

### The units.py module

# **Summary**

#### **Classes**

PhysicalQuantity Provides the base class for handling units throughout PyAnsys Geometry.

#### Constants

UNITS Units manager.

### **PhysicalQuantity**

class ansys.geometry.core.misc.units. $PhysicalQuantity(unit: pint.Unit, expected\_dimensions: pint.Unit | None = None)$ 

Provides the base class for handling units throughout PyAnsys Geometry.

#### **Parameters**

unit

[Unit] Units for the class.

### expected\_dimensions

[Unit, default: None] Units for the dimensionality of the physical quantity.

#### Overview

### **Properties**

unit	Unit of the object.
base_unit	Base unit of the object.

# Import detail

from ansys.geometry.core.misc.units import PhysicalQuantity

# **Property detail**

# **Description**

Provides for handling units homogeneously throughout PyAnsys Geometry.

# Module detail

units.UNITS

Units manager.

# **Description**

Provides the PyAnsys Geometry miscellaneous subpackage.

# The parameters package

# **Summary**

# **Submodules**

parameter Provides get and set methods for parameters.

The parameter.py module

**Summary** 

**Classes** 

Parameter Represents a parameter.

#### **Enums**

ParameterType	Provides values for the parameter types supported.
ParameterUpdateStatus	Provides values for the status messages associated with parameter updates.

#### **Parameter**

Represents a parameter.

### **Parameters**

id

[int] Unique ID for the parameter.

name

[str] Name of the parameter.

dimension\_type

[ParameterType] Type of the parameter.

dimension\_value

[float] Value of the parameter.

### Overview

# **Properties**

name	Get the name of the parameter.
dimension_value	Get the value of the parameter.
dimension_type	Get the type of the parameter.

### **Attributes**

id

# Import detail

from ansys.geometry.core.parameters.parameter import Parameter

# **Property detail**

```
property Parameter.name: str
    Get the name of the parameter.

property Parameter.dimension_value: ansys.geometry.core.typing.Real
    Get the value of the parameter.

property Parameter.dimension_type: ParameterType
    Get the type of the parameter.
```

#### Attribute detail

Parameter.id

### ParameterType

```
class ansys.geometry.core.parameters.parameter.ParameterType(*args, **kwds)
    Bases: enum.Enum
```

Provides values for the parameter types supported.

#### Overview

#### **Attributes**

DIMENSIONTYPE\_UNKNOWN
DIMENSIONTYPE\_LINEAR
DIMENSIONTYPE\_DIAMETRIC
DIMENSIONTYPE\_RADIAL
DIMENSIONTYPE\_ARC
DIMENSIONTYPE\_AREA
DIMENSIONTYPE\_VOLUME
DIMENSIONTYPE\_MASS
DIMENSIONTYPE\_ANGULAR
DIMENSIONTYPE\_COUNT
DIMENSIONTYPE\_UNITLESS

### Import detail

```
from ansys.geometry.core.parameters.parameter import ParameterType
```

#### Attribute detail

```
ParameterType.DIMENSIONTYPE_UNKNOWN = 0

ParameterType.DIMENSIONTYPE_LINEAR = 1

ParameterType.DIMENSIONTYPE_DIAMETRIC = 2

ParameterType.DIMENSIONTYPE_RADIAL = 3

ParameterType.DIMENSIONTYPE_ARC = 4
```

```
ParameterType.DIMENSIONTYPE_AREA = 5

ParameterType.DIMENSIONTYPE_VOLUME = 6

ParameterType.DIMENSIONTYPE_MASS = 7

ParameterType.DIMENSIONTYPE_ANGULAR = 8

ParameterType.DIMENSIONTYPE_COUNT = 9
```

ParameterType.DIMENSIONTYPE\_UNITLESS = 10

# ParameterUpdateStatus

```
\textbf{class} \  \, \text{ansys.geometry.core.parameters.parameter.} \textbf{ParameterUpdateStatus} (*\textit{args}, **\textit{kwds})
```

Bases: enum. Enum

Provides values for the status messages associated with parameter updates.

### Overview

### **Attributes**

SUCCESS
FAILURE
CONSTRAINED\_PARAMETERS
UNKNOWN

# Import detail

from ansys.geometry.core.parameters.parameter import ParameterUpdateStatus

#### **Attribute detail**

```
ParameterUpdateStatus.SUCCESS = 0
ParameterUpdateStatus.FAILURE = 1
ParameterUpdateStatus.CONSTRAINED_PARAMETERS = 2
ParameterUpdateStatus.UNKNOWN = 3
```

# **Description**

Provides get and set methods for parameters.

# **Description**

PyAnsys Geometry parameters subpackage.

## The plotting package

### **Summary**

## **Subpackages**

widgets Submodule providing widgets for the PyAnsys Geometry plotter.

### **Submodules**

plotter Provides plotting for various PyAnsys Geometry objects.

# The widgets package

### **Summary**

### **Submodules**

show\_design\_point Provides the ruler widget for the PyAnsys Geometry plotter.

### The show\_design\_point.py module

### **Summary**

#### **Classes**

*ShowDesignPoints* Provides the a button to hide/show DesignPoint objects in the plotter.

# ShowDesignPoints

 ${\bf class} \ {\bf ansys.geometry.core.plotting.widgets.show\_design\_point.} {\bf ShowDesignPoints} ({\it plotter\_helper:} {\bf class}) {\bf class} \ {\bf ansys.geometry.core.plotting.widgets.show\_design\_point.} {\bf ShowDesignPoints} ({\it plotter\_helper:} {\bf class}) {\bf class} \ {\bf cla$ 

an-

sys.tools.visualization\_interface.

Bases: ansys.tools.visualization\_interface.backends.pyvista.widgets.PlotterWidget

Provides the a button to hide/show DesignPoint objects in the plotter.

#### **Parameters**

# plotter\_helper

[GeometryPlotter] Provides the plotter to add the button to.

### **Overview**

#### Methods

callback	Remove or add the DesignPoint actors upon click.
update	Define the configuration and representation of the button widget.

# **Attributes**

plotter\_helper

### Import detail

from ansys.geometry.core.plotting.widgets.show\_design\_point import ShowDesignPoints

### Attribute detail

ShowDesignPoints.plotter\_helper

### **Method detail**

 $ShowDesignPoints.callback(state: bool) \rightarrow None$ 

Remove or add the DesignPoint actors upon click.

#### **Parameters**

state

[bool] State of the button, which is inherited from PyVista. The value is True if the button is active.

ShowDesignPoints.update()  $\rightarrow$  None

Define the configuration and representation of the button widget.

# **Description**

Provides the ruler widget for the PyAnsys Geometry plotter.

### **Description**

Submodule providing widgets for the PyAnsys Geometry plotter.

The plotter.py module

# **Summary**

Classes

GeometryPlotter Plotter for PyAnsys Geometry objects.

# Constants

POLYDATA\_COLOR\_CYCLER

# GeometryPlotter

**class** ansys.geometry.core.plotting.plotter.**GeometryPlotter**(use\_trame: bool | None = None,

use\_service\_colors: bool | None = None, allow\_picking: bool = False,

 $show\_plane: bool = True$ )

Bases: ansys.tools.visualization\_interface.Plotter

Plotter for PyAnsys Geometry objects.

This class is an implementation of the PlotterInterface class.

#### **Parameters**

#### use trame

[bool, optional] Whether to use trame visualizer or not, by default None.

#### use service colors

[bool, optional] Whether to use service colors or not, by default None.

# allow\_picking

[bool, optional] Whether to allow picking or not, by default False.

# show\_plane

[bool, optional] Whether to show the plane in the scene, by default True.

### Overview

#### Methods

add_frame	Plot a frame in the scene.
add_plane	Plot a plane in the scene.
add_sketch	Plot a sketch in the scene.
add_body_edges	Add the outer edges of a body to the plot.
add_body	Add a body to the scene.
add_face	Add a face to the scene.
add_component	Add a component to the scene.
<pre>add_component_by_body</pre>	Add a component on a per body basis.
add_sketch_polydata	Add sketches to the scene from PyVista polydata.
add_design_point	Add a DesignPoint object to the plotter.
plot_iter	Add a list of any type of object to the scene.
plot	Add a custom mesh to the plotter.
show	Show the plotter.
export_glb	Export the design to a glb file. Does not support picked objects.

# **Properties**

# Import detail

from ansys.geometry.core.plotting.plotter import GeometryPlotter

## **Property detail**

## property GeometryPlotter.use\_service\_colors: bool

Indicates whether to use service colors for plotting purposes.

#### Method detail

GeometryPlotter.add\_frame(frame: ansys.geometry.core.math.frame.Frame, plotting\_options: dict | None = None)  $\rightarrow None$ 

Plot a frame in the scene.

### **Parameters**

#### frame

[Frame] Frame to render in the scene.

## plotting\_options

[dict, default: None] dictionary containing parameters accepted by the pyvista. create\_axes\_marker() class for customizing the frame rendering in the scene.

GeometryPlotter.add\_plane(plane: ansys.geometry.core.math.plane.Plane, plane\_options:  $dict \mid None = None$ , plotting\_options:  $dict \mid None = None$ )  $\rightarrow$  None

Plot a plane in the scene.

### **Parameters**

### plane

[Plane] Plane to render in the scene.

# plane\_options

[dict, default: None] dictionary containing parameters accepted by the pyvista.Plane function for customizing the mesh representing the plane.

### plotting options

[dict, default: None] dictionary containing parameters accepted by the Plotter. add\_mesh method for customizing the mesh rendering of the plane.

GeometryPlotter.add\_sketch(sketch: ansys.geometry.core.sketch.sketch.sketch, show\_plane: bool = False,  $show\_frame: bool = False$ , \*\*plotting\_options:  $dict \mid None$ )  $\rightarrow$  None

Plot a sketch in the scene.

# **Parameters**

### sketch

[Sketch] Sketch to render in the scene.

### show\_plane

[bool, default: False] Whether to render the sketch plane in the scene.

## show frame

[bool, default: False] Whether to show the frame in the scene.

### \*\*plotting options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

GeometryPlotter.add\_body\_edges(body\_plot: ansys.tools.visualization\_interface.MeshObjectPlot, \*\*plotting\_options: dict | None)  $\rightarrow$  None

Add the outer edges of a body to the plot.

This method has the side effect of adding the edges to the GeomObject that you pass through the parameters.

### **Parameters**

## body\_plot

[MeshObjectPlot] Body of which to add the edges.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

GeometryPlotter.add\_body(body: ansys.geometry.core.designer.body.Body, merge: bool = False, \*\* $plotting\_options: dict \mid None$ )  $\rightarrow$  None

Add a body to the scene.

### **Parameters**

### body

[Body] Body to add.

### merge

[bool, default: False] Whether to merge the body into a single mesh. When True, the individual faces of the tessellation are merged. This preserves the number of triangles and only merges the topology.

# \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

 $\label{eq:GeometryPlotter.add_face} Geometry.core.designer.face. Face, **plotting\_options: dict \mid None) \rightarrow \\ None$ 

Add a face to the scene.

## **Parameters**

## face

[Face] Face to add.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

GeometryPlotter.add\_component(component: ansys.geometry.core.designer.component.Component,  $merge\_component: bool = False, merge\_bodies: bool = False, **plotting\_options) \rightarrow None$ 

Add a component to the scene.

### **Parameters**

### component

[Component] Component to add.

### merge\_component

[bool, default: False] Whether to merge the component into a single dataset. When True, all the individual bodies are effectively combined into a single dataset without any hierarchy.

# merge\_bodies

[bool, default: False] Whether to merge each body into a single dataset. When True, all the faces of each individual body are effectively combined into a single dataset without separating faces.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add mesh method.

GeometryPlotter.add\_component\_by\_body(component: ansys.geometry.core.designer.component.Component,  $merge\_bodies: bool, **plotting\_options: dict \mid None) \rightarrow None$ 

Add a component on a per body basis.

#### **Parameters**

### component

[Component] Component to add.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

#### **Notes**

This will allow to make use of the service colors. At the same time, it will be slower than the add\_component method.

GeometryPlotter.add\_sketch\_polydata(polydata\_entries: list[pyvista.PolyData], sketch:

ansys.geometry.core.sketch.sketch.Sketch = None,

\*\* $plotting\_options$ )  $\rightarrow$  None

Add sketches to the scene from PyVista polydata.

### **Parameters**

# polydata\_entries

[list[pyvista.PolyData]] Polydata to add.

### sketch

[Sketch, default: None] Sketch to add.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the  ${\tt Plotter.add\_mesh\ method.}$ 

GeometryPlotter.add\_design\_point( $design\_point$ : ansys.geometry.core.designer.designpoint.DesignPoint, \*\*plotting\_options)  $\rightarrow$  None

Add a DesignPoint object to the plotter.

## **Parameters**

## design\_point

[DesignPoint] DesignPoint to add.

GeometryPlotter.plot\_iter(plotting\_list: list[Any], name\_filter: str = None, \*\*plotting\_options)  $\rightarrow$  None Add a list of any type of object to the scene.

These types of objects are supported: Body, Component, list[pv.PolyData], pv.MultiBlock, and Sketch.

### **Parameters**

### plotting list

[list[Any]] list of objects you want to plot.

## name\_filter

[str, default: None] Regular expression with the desired name or names you want to include in the plotter.

# $**plotting\_options$

[dict, default: None] Keyword arguments. For allowable keyword arguments, see the Plotter.add\_mesh method.

GeometryPlotter.plot(plottable\_object: Any, name\_filter: str = None, \*\*plotting\_options)  $\rightarrow$  None Add a custom mesh to the plotter.

### **Parameters**

### plottable object

[str, default: None] Regular expression with the desired name or names you want to include in the plotter.

### name filter: str, default: None

Regular expression with the desired name or names you want to include in the plotter.

## \*\*plotting\_options

[dict, default: None] Keyword arguments. For allowable keyword arguments, depend of the backend implementation you are using.

GeometryPlotter. **show**( $plotting\_object: Any = None, screenshot: str | None = None, **plotting\_options) <math>\rightarrow$  None | list[Any]

Show the plotter.

### **Parameters**

## plotting\_object

[Any, default: None] Object you can add to the plotter.

### screenshot

[str, default: None] Path to save a screenshot of the plotter.

## \*\*plotting\_options

[dict, default: None] Keyword arguments for the plotter. Arguments depend of the backend implementation you are using.

GeometryPlotter.export\_glb( $plotting\_object$ : Any = None, filename:  $str \mid pathlib.Path \mid None = None$ ,  $**plotting\_options$ )  $\rightarrow$  pathlib.Path

Export the design to a glb file. Does not support picked objects.

## Parameters

## plotting object

[Any, default: None] Object you can add to the plotter.

### filename

[str | Path, default: None] Path to save a GLB file of the plotter. If None, the file will be saved as temp\_glb.glb.

## \*\*plotting\_options

[dict, default: None] Keyword arguments for the plotter. Arguments depend of the backend implementation you are using.

### Returns

### Path

Path to the exported glb file.

## **Description**

Provides plotting for various PyAnsys Geometry objects.

# Module detail

 ${\tt plotter.POLYDATA\_COLOR\_CYCLER}$ 

# **Description**

Provides the PyAnsys Geometry plotting subpackage.

# The shapes package

# **Summary**

# **Subpackages**

curves	Provides the PyAnsys Geometry curves subpackage.
surfaces	Provides the PyAnsys Geometry surface subpackage.

# **Submodules**

box_uv	Provides the BoxUV class.
parameterization	Provides the parametrization-related classes.

# The curves package

# **Summary**

## **Submodules**

circle	Provides for creating and managing a circle.
curve	Provides the Curve class.
curve_evaluation	Provides for creating and managing a curve.
ellipse	Provides for creating and managing an ellipse.
line	Provides for creating and managing a line.
nurbs	Provides for creating and managing a NURBS curve.
trimmed_curve	Trimmed curve class.

# The circle.py module

# **Summary**

# Classes

Circle	Provides 3D circle representation.
CircleEvaluation	Provides evaluation of a circle at a given parameter.

# Circle

class ansys.geometry.core.shapes.curves.circle.Circle(origin: numpy.ndarray |

 $ansys.geometry.core.typing.Real Sequence \mid ansys.geometry.core.math.point.Point3D,\\$ 

radius: pint.Quantity | an-

 $sys.geometry.core.misc.measurements. Distance \\ | \textit{ansys.geometry.core.typing.Real, reference:} \\$ 

numpy.ndarray

 $ansys. geometry. core. typing. Real Sequence \mid$ 

an-

sys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_X, axis: numpy.ndarray | ansys.geometry.core.typing.RealSequence |

sys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_Z)

Bases: ansys.geometry.core.shapes.curves.curve.Curve

Provides 3D circle representation.

### **Parameters**

## origin

[ndarray | RealSequence | Point3D] Origin of the circle.

### radius

[Quantity | Distance | Real] Radius of the circle.

### reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## axis

[ndarray | RealSequence | UnitVector3D | Vector3D] Z-axis direction.

## **Overview**

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the curve.
contains_point	Check a point is contained by the curve.

## **Methods**

evaluate	Evaluate the circle at a given parameter.
transformed_copy	Create a transformed copy of the circle from a transformation matrix.
mirrored_copy	Create a mirrored copy of the circle along the y-axis.
<pre>project_point</pre>	Project a point onto the circle and evauate the circle.
is_coincident_circle	Determine if the circle is coincident with another.
parameterization	Get the parametrization of the circle.

# **Properties**

origin	Origin of the circle.
radius	Radius of the circle.
diameter	Diameter of the circle.
perimeter	Perimeter of the circle.
area	Area of the circle.
dir_x	X-direction of the circle.
dir_y	Y-direction of the circle.
dir_z	Z-direction of the circle.

# **Special methods**

\_\_eq\_\_ Equals operator for the Circle class.

## Import detail

from ansys.geometry.core.shapes.curves.circle import Circle

# **Property detail**

property Circle.origin: ansys.geometry.core.math.point.Point3D

Origin of the circle.

property Circle.radius: pint.Quantity

Radius of the circle.

property Circle.diameter: pint.Quantity

Diameter of the circle.

property Circle.perimeter: pint.Quantity

Perimeter of the circle.

property Circle.area: pint.Quantity

Area of the circle.

property Circle.dir\_x: ansys.geometry.core.math.vector.UnitVector3D

X-direction of the circle.

 $\label{property_core.math.vector.UnitVector3D} \textbf{property}. \textbf{Circle.dir\_y: ansys.geometry.core.math.vector.UnitVector3D}$ 

Y-direction of the circle.

property Circle.dir\_z: ansys.geometry.core.math.vector.UnitVector3D

Z-direction of the circle.

## **Method detail**

 ${\tt Circle.\_\_eq\_\_(\it other: Circle)} \rightarrow {\tt bool}$ 

Equals operator for the Circle class.

 $Circle.evaluate(parameter: ansys.geometry.core.typing.Real) \rightarrow CircleEvaluation$ 

Evaluate the circle at a given parameter.

### **Parameters**

### parameter

[Real] Parameter to evaluate the circle at.

### Returns

## CircleEvaluation

Resulting evaluation.

Circle.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Circle$ 

Create a transformed copy of the circle from a transformation matrix.

### **Parameters**

### matrix

[Matrix44] 4x4 transformation matrix to apply to the circle.

### **Returns**

#### Circle

New circle that is the transformed copy of the original circle.

# Circle.mirrored\_copy() $\rightarrow$ Circle

Create a mirrored copy of the circle along the y-axis.

## Returns

#### Circle

A new circle that is a mirrored copy of the original circle.

Circle.project\_point(point: ansys.geometry.core.math.point3D) → CircleEvaluation

Project a point onto the circle and evauate the circle.

## **Parameters**

# point

[Point3D] Point to project onto the circle.

### **Returns**

## CircleEvaluation

Resulting evaluation.

# ${\tt Circle.is\_coincident\_circle}(\textit{other}: {\tt Circle}) \rightarrow {\tt bool}$

Determine if the circle is coincident with another.

## **Parameters**

### other

[Circle] Circle to determine coincidence with.

## Returns

bool

True if this circle is coincident with the other, False otherwise.

 $Circle.parameterization() \rightarrow ansys.geometry.core.shapes.parameterization.Parameterization$ 

Get the parametrization of the circle.

The parameter of a circle specifies the clockwise angle around the axis (right-hand corkscrew law), with a zero parameter at dir\_x and a period of 2\*pi.

### Returns

### **Parameterization**

Information about how the circle is parameterized.

**abstractmethod** Circle.contains\_param( $param: ansys.geometry.core.typing.Real) <math>\rightarrow$  bool

Check a parameter is within the parametric range of the curve.

**abstractmethod** Circle.contains\_point(point: ansys.geometry.core.math.point.Point3D) → bool

Check a point is contained by the curve.

The point can either lie within the curve or on its boundary.

## CircleEvaluation

Bases: ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation

Provides evaluation of a circle at a given parameter.

### **Parameters**

circle: ~ansys.geometry.core.shapes.curves.circle.Circle

Circle to evaluate.

parameter: Real

Parameter to evaluate the circle at.

### Overview

### **Properties**

circle	Circle being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
tangent	Tangent of the evaluation.
normal	Normal to the circle.
first_derivative	First derivative of the evaluation.
second_derivative	Second derivative of the evaluation.
curvature	Curvature of the circle.

## Import detail

from ansys.geometry.core.shapes.curves.circle import CircleEvaluation

## **Property detail**

property CircleEvaluation.circle: Circle

Circle being evaluated.

property CircleEvaluation.parameter: ansys.geometry.core.typing.Real

Parameter that the evaluation is based upon.

property CircleEvaluation.position: ansys.geometry.core.math.point.Point3D

Position of the evaluation.

## Returns

## Point3D

Point that lies on the circle at this evaluation.

### **Returns**

## UnitVector3D

Tangent unit vector to the circle at this evaluation.

### Returns

### UnitVector3D

Normal unit vector to the circle at this evaluation.

 $\textbf{property} \ \texttt{CircleEvaluation}. \textbf{first\_derivative:} \ \textit{ansys.geometry.core.math.vector.Vector3D}$ 

First derivative of the evaluation.

The first derivative is in the direction of the tangent and has a magnitude equal to the velocity (rate of change of position) at that point.

### Returns

### Vector3D

First derivative of the evaluation.

## Returns

### Vector3D

Second derivative of the evaluation.

property CircleEvaluation.curvature: ansys.geometry.core.typing.Real

Curvature of the circle.

## Returns

### Real

Curvature of the circle.

# **Description**

Provides for creating and managing a circle.

The curve.py module

## **Summary**

### **Classes**

*Curve* Provides the abstract base class representing a 3D curve.

### Curve

class ansys.geometry.core.shapes.curves.curve.Curve

Bases: abc.ABC

Provides the abstract base class representing a 3D curve.

### Overview

### **Abstract methods**

parameterization	Parameterize the curve.
contains_param	Check a parameter is within the parametric range of the curve.
contains_point	Check a point is contained by the curve.
transformed_copy	Create a transformed copy of the curve.
eq	Determine if two curves are equal.
evaluate	Evaluate the curve at the given parameter.
<pre>project_point</pre>	Project a point to the curve.

### Methods

trim Trim this curve by bounding it with an interval.

## Import detail

from ansys.geometry.core.shapes.curves.curve import Curve

# **Method detail**

 $abstractmethod Curve.parameterization() \rightarrow$ 

ansys. geometry. core. shapes. parameterization. Parameterization

Parameterize the curve.

 $\textbf{abstractmethod} \ \ \textbf{Curve.contains\_param}(\textit{param: ansys.geometry.core.typing.Real}) \rightarrow \textbf{bool}$ 

Check a parameter is within the parametric range of the curve.

 $\textbf{abstractmethod} \ \ \textbf{Curve.contains\_point} (\textit{point:} \ \ ansys.geometry.core.math.point.Point3D) \rightarrow bool$ 

Check a point is contained by the curve.

The point can either lie within the curve or on its boundary.

**abstractmethod** Curve.**transformed\_copy**(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Curve$  Create a transformed copy of the curve.

**abstractmethod** Curve. $\_$ **eq** $\_$ (other: Curve)  $\rightarrow$  bool

Determine if two curves are equal.

abstractmethod Curve.evaluate(parameter: ansys.geometry.core.typing.Real)  $\rightarrow$ 

 $ansys. geometry. core. shapes. curve\_evaluation. Curve Evaluation$ 

Evaluate the curve at the given parameter.

**abstractmethod** Curve.project\_point(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow$  ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation

Project a point to the curve.

This method returns the evaluation at the closest point.

Curve.trim(interval: ansys.geometry.core.shapes.parameterization.Interval)  $\rightarrow$  ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve

Trim this curve by bounding it with an interval.

## Returns

## TrimmedCurve

The resulting bounded curve.

# **Description**

Provides the Curve class.

The curve\_evaluation.py module

# **Summary**

### **Classes**

CurveEvaluation	Provides for evaluating a curve.
-----------------	----------------------------------

# CurveEvaluation

Provides for evaluating a curve.

# **Overview**

# **Methods**

*is\_set* Determine if the parameter for the evaluation has been set.

## **Properties**

parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
first_derivative	First derivative of the evaluation.
second_derivative	Second derivative of the evaluation.
curvature	Curvature of the evaluation.

## Import detail

from ansys.geometry.core.shapes.curves.curve\_evaluation import CurveEvaluation

# **Property detail**

property CurveEvaluation.parameter: ansys.geometry.core.typing.Real

#### Abstractmethod

Parameter that the evaluation is based upon.

property CurveEvaluation.position: ansys.geometry.core.math.point.Point3D

### Abstractmethod

Position of the evaluation.

property CurveEvaluation.first\_derivative: ansys.geometry.core.math.vector.Vector3D

## Abstractmethod

First derivative of the evaluation.

property CurveEvaluation.second\_derivative: ansys.geometry.core.math.vector.Vector3D

## Abstractmethod

Second derivative of the evaluation.

property CurveEvaluation.curvature: ansys.geometry.core.typing.Real

### Abstractmethod

Curvature of the evaluation.

### **Method detail**

CurveEvaluation.is\_set()  $\rightarrow$  bool

Determine if the parameter for the evaluation has been set.

# **Description**

Provides for creating and managing a curve.

# The ellipse.py module

# **Summary**

## **Classes**

Ellipse	Provides 3D ellipse representation.
EllipseEvaluation	Evaluate an ellipse at a given parameter.

## **Ellipse**

```
class ansys.geometry.core.shapes.curves.ellipse.Ellipse(origin: numpy.ndarray |
                                                                    ansys.geometry.core.typing.RealSequence |
                                                                    ansys.geometry.core.math.point.Point3D,
                                                                    major_radius: pint.Quantity | an-
                                                                    sys.geometry.core.misc.measurements.Distance
                                                                    | ansys.geometry.core.typing.Real,
                                                                    minor_radius: pint.Quantity | an-
                                                                    sys.geometry.core.misc.measurements.Distance
                                                                    | ansys.geometry.core.typing.Real,
                                                                    reference: numpy.ndarray |
                                                                    ansys.geometry.core.typing.RealSequence |
                                                                    sys.geometry.core.math.vector.UnitVector3D
                                                                    ansys.geometry.core.math.vector.Vector3D
                                                                    = UNITVECTOR3D_X, axis:
                                                                    numpy.ndarray |
                                                                    ansys.geometry.core.typing.RealSequence |
                                                                    sys.geometry.core.math.vector.UnitVector3D
                                                                    ansys.geometry.core.math.vector.Vector 3D\\
                                                                    = UNITVECTOR3D_Z)
```

Bases: ansys.geometry.core.shapes.curves.curve.Curve

Provides 3D ellipse representation.

### **Parameters**

### origin

[ndarray | RealSequence | Point3D] Origin of the ellipse.

## major\_radius

[Quantity | Distance | Real] Major radius of the ellipse.

# $minor\_radius$

[Quantity | Distance | Real] Minor radius of the ellipse.

### reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## axis

[ndarray | RealSequence | UnitVector3D | Vector3D] Z-axis direction.

# **Overview**

### **Abstract methods**

contains_param	Check a parameter is within the parametric range of the curve.
contains_point	Check a point is contained by the curve.

# Methods

mirrored_copy	Create a mirrored copy of the ellipse along the y-axis.
evaluate	Evaluate the ellipse at the given parameter.
<pre>project_point</pre>	Project a point onto the ellipse and evaluate the ellipse.
is_coincident_ellipse	Determine if this ellipse is coincident with another.
transformed_copy	Create a transformed copy of the ellipse from a transformation matrix.
parameterization	Get the parametrization of the ellipse.

# **Properties**

origin	Origin of the ellipse.
major_radius	Major radius of the ellipse.
minor_radius	Minor radius of the ellipse.
dir_x	X-direction of the ellipse.
dir_y	Y-direction of the ellipse.
dir_z	Z-direction of the ellipse.
eccentricity	Eccentricity of the ellipse.
linear_eccentricity	Linear eccentricity of the ellipse.
semi_latus_rectum	Semi-latus rectum of the ellipse.
perimeter	Perimeter of the ellipse.
area	Area of the ellipse.

# **Special methods**

eq	Equals operator for the Ellipse class.
	Equals operator for the ETTTPSC class.

# Import detail

```
from ansys.geometry.core.shapes.curves.ellipse import Ellipse
```

# **Property detail**

```
property Ellipse.origin: ansys.geometry.core.math.point.Point3D
   Origin of the ellipse.
```

```
property Ellipse.major_radius: pint.Quantity
```

Major radius of the ellipse.

property Ellipse.minor\_radius: pint.Quantity

Minor radius of the ellipse.

property Ellipse.dir\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-direction of the ellipse.

property Ellipse.dir\_y: ansys.geometry.core.math.vector.UnitVector3D
 Y-direction of the ellipse.

property Ellipse.dir\_z: ansys.geometry.core.math.vector.UnitVector3D
 Z-direction of the ellipse.

```
property Ellipse.eccentricity: ansys.geometry.core.typing.Real
     Eccentricity of the ellipse.
property Ellipse.linear_eccentricity: pint.Quantity
     Linear eccentricity of the ellipse.
     Notes
     The linear eccentricity is the distance from the center to the focus.
property Ellipse.semi_latus_rectum: pint.Quantity
     Semi-latus rectum of the ellipse.
property Ellipse.perimeter: pint.Quantity
     Perimeter of the ellipse.
property Ellipse.area: pint.Quantity
     Area of the ellipse.
Method detail
Ellipse.__eq__(other: Ellipse) \rightarrow bool
     Equals operator for the Ellipse class.
Ellipse.mirrored_copy() \rightarrow Ellipse
     Create a mirrored copy of the ellipse along the y-axis.
           Returns
               Ellipse
                   New ellipse that is a mirrored copy of the original ellipse.
Ellipse.evaluate(parameter: ansys.geometry.core.typing.Real) \rightarrow EllipseEvaluation
     Evaluate the ellipse at the given parameter.
           Parameters
               parameter
                   [Real] Parameter to evaluate the ellipse at.
           Returns
               EllipseEvaluation
                   Resulting evaluation.
Ellipse.project_point(point: ansys.geometry.core.math.point.Point3D) \rightarrow EllipseEvaluation
     Project a point onto the ellipse and evaluate the ellipse.
           Parameters
               point
                   [Point3D] Point to project onto the ellipse.
           Returns
               EllipseEvaluation
```

Resulting evaluation.

# Ellipse.is\_coincident\_ellipse(other: Ellipse) $\rightarrow$ bool

Determine if this ellipse is coincident with another.

## **Parameters**

#### other

[Ellipse] Ellipse to determine coincidence with.

### Returns

bool

True if this ellipse is coincident with the other, False otherwise.

Ellipse.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Ellipse$ 

Create a transformed copy of the ellipse from a transformation matrix.

#### **Parameters**

### matrix

[Matrix44] 4x4 transformation matrix to apply to the ellipse.

### Returns

## **Ellipse**

New ellipse that is the transformed copy of the original ellipse.

 ${\tt Ellipse.parameterization()} \rightarrow \textit{ansys.geometry.core.shapes.parameterization.} Parameterization$ 

Get the parametrization of the ellipse.

The parameter of an ellipse specifies the clockwise angle around the axis (right-hand corkscrew law), with a zero parameter at dir\_x and a period of 2\*pi.

## Returns

## Parameterization

Information about how the ellipse is parameterized.

 $\textbf{abstractmethod} \ \ \textbf{Ellipse.contains\_param}(\textit{param: ansys.geometry.core.typing.Real}) \rightarrow \textbf{bool}$ 

Check a parameter is within the parametric range of the curve.

**abstractmethod** Ellipse.contains\_point(point: ansys.geometry.core.math.point.Point3D) → bool

Check a point is contained by the curve.

The point can either lie within the curve or on its boundary.

## **EllipseEvaluation**

class ansys.geometry.core.shapes.curves.ellipse.EllipseEvaluation(ellipse: Ellipse, parameter:

an-

sys.geometry.core.typing.Real)

Bases: ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation

Evaluate an ellipse at a given parameter.

### **Parameters**

ellipse: ~ansys.geometry.core.shapes.curves.ellipse.Ellipse

Ellipse to evaluate.

## parameter: float, int

Parameter to evaluate the ellipse at.

# **Overview**

## **Properties**

ellipse	Ellipse being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
tangent	Tangent of the evaluation.
normal	Normal of the evaluation.
first_derivative	Girst derivative of the evaluation.
second_derivative	Second derivative of the evaluation.
curvature	Curvature of the ellipse.

## Import detail

from ansys.geometry.core.shapes.curves.ellipse import EllipseEvaluation

## **Property detail**

property EllipseEvaluation.ellipse: Ellipse

Ellipse being evaluated.

property EllipseEvaluation.parameter: ansys.geometry.core.typing.Real

Parameter that the evaluation is based upon.

property EllipseEvaluation.position: ansys.geometry.core.math.point.Point3D

Position of the evaluation.

### **Returns**

### Point3D

Point that lies on the ellipse at this evaluation.

## Returns

## UnitVector3D

Tangent unit vector to the ellipse at this evaluation.

### Returns

### UnitVector3D

Normal unit vector to the ellipse at this evaluation.

property EllipseEvaluation.first\_derivative: ansys.geometry.core.math.vector.Vector3D
 Girst derivative of the evaluation.

The first derivative is in the direction of the tangent and has a magnitude equal to the velocity (rate of change of position) at that point.

### **Returns**

### Vector3D

First derivative of the evaluation.

property EllipseEvaluation.second\_derivative: ansys.geometry.core.math.vector.Vector3D Second derivative of the evaluation.

### Returns

## Vector3D

Second derivative of the evaluation.

property EllipseEvaluation.curvature: ansys.geometry.core.typing.Real
 Curvature of the ellipse.

#### Returns

#### Real

Curvature of the ellipse.

# **Description**

Provides for creating and managing an ellipse.

# The line.py module

## **Summary**

## **Classes**

Line	Provides 3D line representation.
LineEvaluation	Provides for evaluating a line.

### Line

class ansys.geometry.core.shapes.curves.line.Line(origin: numpy.ndarray |

ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.point.Point3D, direction:
numpy.ndarray |
ansys.geometry.core.typing.RealSequence |
ansys.geometry.core.math.vector.UnitVector3D |

ansys.geometry.core.math.vector.Vector3D)

Bases: ansys.geometry.core.shapes.curves.curve.Curve

Provides 3D line representation.

### **Parameters**

### origin

[ndarray | RealSequence | Point3D] Origin of the line.

## direction

[ndarray | RealSequence | UnitVector3D | Vector3D] Direction of the line.

# **Overview**

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the curve.
contains_point	Check a point is contained by the curve.

### Methods

evaluate	Evaluate the line at a given parameter.
transformed_copy	Create a transformed copy of the line from a transformation matrix.
<pre>project_point</pre>	Project a point onto the line and evaluate the line.
is_coincident_line	Determine if the line is coincident with another line.
is_opposite_line	Determine if the line is opposite another line.
parameterization	Get the parametrization of the line.

# **Properties**

origin	Origin of the line.
direction	Direction of the line.

# **Special methods**

# Import detail

from ansys.geometry.core.shapes.curves.line import Line

# **Property detail**

```
property Line.origin: ansys.geometry.core.math.point.Point3D
    Origin of the line.
```

property Line.direction: ansys.geometry.core.math.vector.UnitVector3D
 Direction of the line.

## Method detail

```
Line.__eq__(other: object) \rightarrow bool
Equals operator for the Line class.
```

Line.evaluate(parameter: float)  $\rightarrow$  LineEvaluation

Evaluate the line at a given parameter.

## **Parameters**

# parameter

[Real] Parameter to evaluate the line at.

### **Returns**

### LineEvaluation

Resulting evaluation.

Line.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Line$ 

Create a transformed copy of the line from a transformation matrix.

### **Parameters**

#### matrix

[Matrix44] 4X4 transformation matrix to apply to the line.

### **Returns**

Line

New line that is the transformed copy of the original line.

Line.project\_point(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow$  LineEvaluation

Project a point onto the line and evaluate the line.

### **Parameters**

point

[Point3D] Point to project onto the line.

### Returns

### LineEvaluation

Resulting evaluation.

# Line.is\_coincident\_line(other: Line) $\rightarrow bool$

Determine if the line is coincident with another line.

## **Parameters**

other

[Line] Line to determine coincidence with.

### **Returns**

bool

True if the line is coincident with another line, False otherwise.

# Line.is\_opposite\_line(other: Line) $\rightarrow bool$

Determine if the line is opposite another line.

## **Parameters**

other

[Line] Line to determine opposition with.

## Returns

bool

True if the line is opposite to another line.

Line.parameterization()  $\rightarrow$  ansys.geometry.core.shapes.parameterization.Parameterization

Get the parametrization of the line.

The parameter of a line specifies the distance from the *origin* in the direction of *direction*.

### Returns

### **Parameterization**

Information about how the line is parameterized.

**abstractmethod** Line.contains\_param(param: ansys.geometry.core.typing.Real) → bool

Check a parameter is within the parametric range of the curve.

**abstractmethod** Line.contains\_point(point: ansys.geometry.core.math.point.Point3D) → bool

Check a point is contained by the curve.

The point can either lie within the curve or on its boundary.

## LineEvaluation

class ansys.geometry.core.shapes.curves.line.LineEvaluation(line: Line, parameter: float = None)
 Bases: ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation
 Provides for evaluating a line.

### Overview

## **Properties**

line	Line being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
tangent	Tangent of the evaluation.
first_derivative	First derivative of the evaluation.
second_derivative	Second derivative of the evaluation.
curvature	Curvature of the line, which is always <b>0</b> .

# Import detail

from ansys.geometry.core.shapes.curves.line import LineEvaluation

# **Property detail**

property LineEvaluation.line: Line

Line being evaluated.

property LineEvaluation.parameter: float

Parameter that the evaluation is based upon.

property LineEvaluation.position: ansys.geometry.core.math.point.Point3D

Position of the evaluation.

### Returns

### Point3D

Point that lies on the line at this evaluation.

### Returns

### UnitVector3D

Tangent unit vector to the line at this evaluation.

## **Notes**

This is always equal to the direction of the line.

# property LineEvaluation.first\_derivative: ansys.geometry.core.math.vector.Vector3D

First derivative of the evaluation.

The first derivative is always equal to the direction of the line.

### **Returns**

### Vector3D

First derivative of the evaluation.

# property LineEvaluation.second\_derivative: ansys.geometry.core.math.vector.Vector3D

Second derivative of the evaluation.

The second derivative is always equal to a zero vector Vector3D([0, 0, 0]).

#### Returns

#### Vector3D

Second derivative of the evaluation, which is always Vector3D([0, 0, 0]).

## property LineEvaluation.curvature: float

Curvature of the line, which is always 0.

### **Returns**

#### Real

Curvature of the line, which is always 0.

# **Description**

Provides for creating and managing a line.

## The nurbs.py module

## **Summary**

### **Classes**

NURBSCurve	Represents a NURBS curve.
NURBSCurveEvaluation	Provides evaluation of a NURBS curve at a given parameter.

## **NURBSCurve**

class ansys.geometry.core.shapes.curves.nurbs.NURBSCurve

Bases: ansys.geometry.core.shapes.curves.curve.Curve

Represents a NURBS curve.

### **Notes**

This class is a wrapper around the NURBS curve class from the *geomdl* library. By leveraging the *geomdl* library, this class provides a high-level interface to create and manipulate NURBS curves. The *geomdl* library is a powerful library for working with NURBS curves and surfaces. For more information, see https://pypi.org/project/geomdl/.

# **Overview**

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the curve.
contains_point	Check a point is contained by the curve.

## **Constructors**

# Methods

parameterization	Get the parametrization of the NURBS curve.
transformed_copy	Create a transformed copy of the curve.
evaluate	Evaluate the curve at the given parameter.
<pre>project_point</pre>	Project a point to the NURBS curve.

# **Properties**

geomdl_nurbs_curve	Get the underlying NURBS curve.
control_points	Get the control points of the curve.
degree	Get the degree of the curve.
knots	Get the knot vector of the curve.
weights	Get the weights of the control points.

# **Special methods**

\_\_eq\_\_ Determine if two curves are equal.

# Import detail

from ansys.geometry.core.shapes.curves.nurbs import NURBSCurve

# **Property detail**

property NURBSCurve.geomdl\_nurbs\_curve: geomdl.NURBS.Curve

Get the underlying NURBS curve.

## **Notes**

This property gives access to the full functionality of the NURBS curve coming from the *geomdl* library. Use with caution.

property NURBSCurve.control\_points: list[ansys.geometry.core.math.Point3D]

Get the control points of the curve.

```
property NURBSCurve.degree: int
     Get the degree of the curve.
property NURBSCurve.knots: list[ansys.geometry.core.typing.Real]
     Get the knot vector of the curve.
property NURBSCurve.weights: list[ansys.geometry.core.typing.Real]
     Get the weights of the control points.
Method detail
classmethod NURBSCurve.from_control_points(control_points: list[ansys.geometry.core.math.Point3D],
                                                    degree: int, knots: list[ansys.geometry.core.typing.Real],
                                                    weights: list[ansys.geometry.core.typing.Real] = None) \rightarrow
                                                    NURBSCurve
     Create a NURBS curve from control points.
           Parameters
               control_points
                   [list[Point3D]] Control points of the curve.
               degree
                   [int] Degree of the curve.
               knots
                   [list[Real]] Knot vector of the curve.
               weights
                   [list[Real], optional] Weights of the control points.
           Returns
               NURBSCurve
                   NURBS curve.
NURBSCurve.\_eq\_(other: NURBSCurve) \rightarrow bool
     Determine if two curves are equal.
NURBSCurve.parameterization() \rightarrow ansys.geometry.core.shapes.parameterization.Parameterization
     Get the parametrization of the NURBS curve.
     The parameter is defined in the interval [0, 1] by default. Information is provided about the parameter type and
     form.
           Returns
               Parameterization
                   Information about how the NURBS curve is parameterized.
NURBSCurve.transformed_copy(matrix: ansys.geometry.core.math.Matrix44) \rightarrow NURBSCurve
     Create a transformed copy of the curve.
           Parameters
               matrix
                   [Matrix44] Transformation matrix.
           Returns
```

**NURBSCurve** 

Transformed copy of the curve.

 $\label{eq:nurses} \begin{tabular}{ll} NURBSCurve.evaluate(parameter: ansys.geometry.core.typing.Real) \rightarrow \\ ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation.\\ \end{tabular}$ 

Evaluate the curve at the given parameter.

#### **Parameters**

### parameter

[Real] Parameter to evaluate the curve at.

### Returns

### CurveEvaluation

Evaluation of the curve at the given parameter.

abstractmethod NURBSCurve.contains\_param(param: ansys.geometry.core.typing.Real)  $\rightarrow$  bool Check a parameter is within the parametric range of the curve.

 $\textbf{abstractmethod} \ \ \textbf{NURBSCurve.contains\_point}(\textit{point: ansys.geometry.core.math.Point3D}) \rightarrow \textbf{bool}$  Check a point is contained by the curve.

The point can either lie within the curve or on its boundary.

```
NURBSCurve.project_point(point: ansys.geometry.core.math.Point3D, initial_guess: ansys.geometry.core.typing.Real | None = None) \rightarrow ansys.geometry.core.shapes.curves.curve_evaluation.CurveEvaluation
```

Project a point to the NURBS curve.

This method returns the evaluation at the closest point.

## **Parameters**

## point

[Point3D] Point to project to the curve.

### initial guess

[Real, optional] Initial guess for the optimization algorithm. If not provided, the midpoint of the domain is used.

## Returns

### CurveEvaluation

Evaluation at the closest point on the curve.

# **Notes**

Based on the NURBS book, the projection of a point to a NURBS curve is the solution to the following optimization problem: minimize the distance between the point and the curve. The distance is defined as the Euclidean distance squared. For more information, please refer to the implementation of the *distance\_squared* function.

## NURBSCurveEvaluation

Bases: ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation

Provides evaluation of a NURBS curve at a given parameter.

## **Parameters**

**nurbs\_curve:** ~ansys.geometry.core.shapes.curves.nurbs.NURBSCurve NURBS curve to evaluate.

parameter: Real

Parameter to evaluate the NURBS curve at.

### Overview

## **Properties**

parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
first_derivative	First derivative of the evaluation.
second_derivative	Second derivative of the evaluation.
curvature	Curvature of the evaluation.

# Import detail

from ansys.geometry.core.shapes.curves.nurbs import NURBSCurveEvaluation

# **Property detail**

property NURBSCurveEvaluation.parameter: ansys.geometry.core.typing.Real
 Parameter that the evaluation is based upon.

property NURBSCurveEvaluation.position: ansys.geometry.core.math.Point3D
 Position of the evaluation.

property NURBSCurveEvaluation.first\_derivative: ansys.geometry.core.math.vector.Vector3D
 First derivative of the evaluation.

## **Description**

Provides for creating and managing a NURBS curve.

# The trimmed\_curve.py module

# **Summary**

# Classes

TrimmedCurve	Represents a trimmed curve.
ReversedTrimmedCurve	Represents a reversed trimmed curve.

### TrimmedCurve

class ansys.geometry.core.shapes.curves.trimmed\_curve.TrimmedCurve(geometry: an-

sys.geometry.core.shapes.curves.curve.Curve,

start: an-

sys.geometry.core.math.point.Point3D,

end: an-

sys.geometry.core.math.point.Point3D,

interval: an-

sys.geometry.core.shapes.parameterization.Inter

length: pint.Quantity,

grpc\_client: an-

sys.geometry.core.connection.client.GrpcClient = *None*)

Represents a trimmed curve.

A trimmed curve is a curve that has a boundary. This boundary comes in the form of an interval.

### **Parameters**

### geometry

[Curve] Underlying mathematical representation of the curve.

## start

[Point3D] Start point of the curve.

### end

[Point3D] End point of the curve.

### interval

[Interval] Parametric interval that bounds the curve.

### length

[Quantity] Length of the curve.

## grpc\_client

[GrpcClient, default: None] gRPC client that is required for advanced functions such as intersect\_curve().

# **Overview**

## **Methods**

evaluate_proportion	Evaluate the curve at a proportional value.
intersect_curve	Get the intersect points of this trimmed curve with another one.

# **Properties**

geometry	Underlying mathematical curve.
start	Start point of the curve.
end	End point of the curve.
length	Calculated length of the edge.
interval	Interval of the curve that provides its boundary.

## Special methods

\_\_repr\_\_ Represent the trimmed curve as a string.

## Import detail

from ansys.geometry.core.shapes.curves.trimmed\_curve import TrimmedCurve

# **Property detail**

```
property TrimmedCurve.start: ansys.geometry.core.math.point.Point3D
    Start point of the curve.
```

```
property TrimmedCurve.end: ansys.geometry.core.math.point.Point3D
End point of the curve.
```

```
property TrimmedCurve.length: pint.Quantity
    Calculated length of the edge.
```

### **Method detail**

 $\label{lem:core.evaluate_proportion} \begin{picture}(param: ansys.geometry.core.typing.Real) \to \\ ansys.geometry.core.shapes.curves.curve\_evaluation.CurveEvaluation$ 

Evaluate the curve at a proportional value.

A parameter of 0 corresponds to the start of the curve, while a parameter of 1 corresponds to the end of the curve.

## **Parameters**

## param

[Real] Parameter in the proportional range [0,1].

### Returns

# CurveEvaluation

Resulting curve evaluation.

TrimmedCurve.intersect\_curve(other: TrimmedCurve)  $\rightarrow$  list[ansys.geometry.core.math.point.Point3D] Get the intersect points of this trimmed curve with another one.

If the two trimmed curves do not intersect, an empty list is returned.

### **Parameters**

# other

[TrimmedCurve] Trimmed curve to intersect with.

### Returns

### list[Point3D]

All points of intersection between the curves.

```
TrimmedCurve.__repr__() \rightarrow str
```

Represent the trimmed curve as a string.

### ReversedTrimmedCurve

class ansys.geometry.core.shapes.curves.trimmed\_curve.ReversedTrimmedCurve(geometry: an-

sys.geometry.core.shapes.curves.curv

start: an-

sys.geometry.core.math.point.Point31

end: an-

sys.geometry.core.math.point.Point3l

interval: an-

sys.geometry.core.shapes.parameteriz

length:

pint.Quantity,

grpc\_client: an-

sys.geometry.core.connection.client.

= None)

Bases: TrimmedCurve

Represents a reversed trimmed curve.

When a curve is reversed, its start and end points are swapped, and parameters for evaluations are handled to provide expected results conforming to the direction of the curve. For example, evaluating a trimmed curve proportionally at 0 evaluates at the start point of the curve, but evaluating a reversed trimmed curve proportionally at 0 evaluates at what was previously the end point of the curve but is now the start point.

# **Parameters**

### geometry

[Curve] Underlying mathematical representation of the curve.

### start

[Point3D] Original start point of the curve.

### end

[Point3D] Original end point of the curve.

### interval

[Interval] Parametric interval that bounds the curve.

# length

[Quantity] Length of the curve.

# grpc\_client

[GrpcClient, default: None] gRPC client that is required for advanced functions such as intersect\_curve().

### Overview

### **Methods**

# Import detail

from ansys.geometry.core.shapes.curves.trimmed\_curve import ReversedTrimmedCurve

## **Method detail**

 $Reversed Trimmed Curve. \textbf{evaluate\_proportion} (param: ansys. geometry. core. typing. Real) \rightarrow ansys. geometry. core. shapes. curves. curve\_evaluation. Curve Evaluation$ 

Evaluate the curve at a proportional value.

A parameter of 0 corresponds to the start of the curve, while a parameter of 1 corresponds to the end of the curve.

### **Parameters**

## param

[Real] Parameter in the proportional range [0,1].

## **Returns**

### CurveEvaluation

Resulting curve evaluation.

# **Description**

Trimmed curve class.

# **Description**

Provides the PyAnsys Geometry curves subpackage.

# The surfaces package

# **Summary**

## **Submodules**

cone	Provides for creating and managing a cone.
cylinder	Provides for creating and managing a cylinder.
plane	Provides for creating and managing a plane.
sphere	Provides for creating and managing a sphere.
surface	Provides the Surface class.
surface_evaluation	Provides for evaluating a surface.
torus	Provides for creating and managing a torus.
trimmed_surface	Provides the TrimmedSurface class.

# The cone.py module

## **Summary**

## Classes

Cone	Provides 3D cone representation.
ConeEvaluation	Evaluate the cone at given parameters.

### Cone

class ansys.geometry.core.shapes.surfaces.cone.Cone(origin: numpy.ndarray |

ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.point.Point3D, radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, half\_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, reference: numpy.ndarray | ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_X, axis: numpy.ndarray | ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_Z)

Bases: ansys.geometry.core.shapes.surfaces.surface.Surface Provides 3D cone representation.

### **Parameters**

## origin

[ndarray | RealSequence | Point3D] Origin of the cone.

### radius

[Quantity | Distance | Real] Radius of the cone.

## half angle

[Quantity | Angle | Real] Half angle of the apex, determining the upward angle.

## reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

### axis

[ndarray | RealSequence | UnitVector3D | Vector3D] Z-axis direction.

### Overview

### **Abstract methods**

contains_param	Check a parameter is within the parametric range of the surface.
contains_point	Check a point is contained by the surface.

### **Methods**

transformed_copy	Create a transformed copy of the cone from a transformation matrix.
mirrored_copy	Create a mirrored copy of the cone along the y-axis.
evaluate	Evaluate the cone at given parameters.
<pre>project_point</pre>	Project a point onto the cone and evaluate the cone.
parameterization	Parameterize the cone surface as a tuple (U and V respectively).

## **Properties**

origin	Origin of the cone.
radius	Radius of the cone.
half_angle	Half angle of the apex.
dir_x	X-direction of the cone.
dir_y	Y-direction of the cone.
dir_z	Z-direction of the cone.
height	Height of the cone.
surface_area	Surface area of the cone.
volume	Volume of the cone.
apex	Apex point of the cone.
apex_param	Apex parameter of the cone.

# **Special methods**

eq	Equals operator for the Cone class.
----	-------------------------------------

## Import detail

```
from ansys.geometry.core.shapes.surfaces.cone import Cone
```

# **Property detail**

```
property Cone.origin: ansys.geometry.core.math.point.Point3D
     Origin of the cone.
property Cone.radius: pint.Quantity
     Radius of the cone.
property Cone.half_angle: pint.Quantity
     Half angle of the apex.
property Cone.dir_x: ansys.geometry.core.math.vector.UnitVector3D
     X-direction of the cone.
property Cone.dir_y: ansys.geometry.core.math.vector.UnitVector3D
     Y-direction of the cone.
property Cone.dir_z: ansys.geometry.core.math.vector.UnitVector3D
     Z-direction of the cone.
property Cone.height: pint.Quantity
     Height of the cone.
property Cone.surface_area: pint.Quantity
     Surface area of the cone.
property Cone.volume: pint.Quantity
     Volume of the cone.
property Cone.apex: ansys.geometry.core.math.point.Point3D
     Apex point of the cone.
```

### property Cone.apex\_param: ansys.geometry.core.typing.Real

Apex parameter of the cone.

### Method detail

Cone.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Cone$ 

Create a transformed copy of the cone from a transformation matrix.

#### **Parameters**

#### matrix

[Matrix44] 4x4 transformation matrix to apply to the cone.

#### Returns

Cone

New cone that is the transformed copy of the original cone.

## Cone.mirrored\_copy() $\rightarrow$ *Cone*

Create a mirrored copy of the cone along the y-axis.

### **Returns**

Cone

New cone that is a mirrored copy of the original cone.

Cone.**\_\_eq\_\_**(other: Cone) 
$$\rightarrow$$
 bool

Equals operator for the Cone class.

 $\textbf{Cone.evaluate}(\textit{parameter}: ansys.geometry.core.shapes.parameterization.ParamUV) \rightarrow \textit{ConeEvaluation}$ 

Evaluate the cone at given parameters.

## **Parameters**

### parameter

[ParamUV] Parameters (u,v) to evaluate the cone at.

### Returns

# ConeEvaluation

Resulting evaluation.

 $\texttt{Cone.project\_point}(\textit{point:} \ ansys. \textit{geometry.core.math.point.Point3D}) \rightarrow \textit{ConeEvaluation}$ 

Project a point onto the cone and evaluate the cone.

### **Parameters**

point

[Point3D] Point to project onto the cone.

## Returns

## ConeEvaluation

Resulting evaluation.

Cone. parameterization()  $\rightarrow$  tuple[ansys.geometry.core.shapes.parameterization.Parameterization, ansys.geometry.core.shapes.parameterization.Parameterization]

Parameterize the cone surface as a tuple (U and V respectively).

The U parameter specifies the clockwise angle around the axis (right-hand corkscrew law), with a zero parameter at dir\_x and a period of 2\*pi.

The V parameter specifies the distance along the axis, with a zero parameter at the XY plane of the cone.

### Returns

## tuple[Parameterization, Parameterization]

Information about how a cone's u and v parameters are parameterized, respectively.

 $\begin{tabular}{ll} \textbf{abstractmethod} & \textbf{Cone.contains\_param}(\textit{param\_uv:} \ ansys.geometry.core.shapes.parameterization.ParamUV) \\ &\rightarrow bool \end{tabular}$ 

Check a parameter is within the parametric range of the surface.

**abstractmethod** Cone.contains\_point(point: ansys.geometry.core.math.point.Point3D) → bool

Check a point is contained by the surface.

The point can either lie within the surface or on its boundary.

## ConeEvaluation

class ansys.geometry.core.shapes.surfaces.cone.ConeEvaluation(cone: Cone, parameter: an-

sys.geometry.core.shapes.parameterization.ParamUV)

Bases: ansys.geometry.core.shapes.surfaces.surface\_evaluation.SurfaceEvaluation

Evaluate the cone at given parameters.

### **Parameters**

cone: ~ansys.geometry.core.shapes.surfaces.cone.Cone

Cone to evaluate.

parameter: ParamUV

Pparameters (u, v) to evaluate the cone at.

# **Overview**

# **Properties**

cone	Cone being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
normal	Normal to the surface.
u_derivative	First derivative with respect to the U parameter.
v_derivative	First derivative with respect to the V parameter.
uu_derivative	Second derivative with respect to the U parameter.
uv_derivative	Second derivative with respect to the U and V parameters.
vv_derivative	Second derivative with respect to the V parameter.
min_curvature	Minimum curvature of the cone.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature of the cone.
max_curvature_direction	Maximum curvature direction.

# Import detail

from ansys.geometry.core.shapes.surfaces.cone import ConeEvaluation

# **Property detail**

property ConeEvaluation.cone: Cone

Cone being evaluated.

property ConeEvaluation.parameter: ansys.geometry.core.shapes.parameterization.ParamUV Parameter that the evaluation is based upon.

property ConeEvaluation.position: ansys.geometry.core.math.point.Point3D
 Position of the evaluation.

#### Returns

### Point3D

Point that lies on the cone at this evaluation.

property ConeEvaluation.normal: ansys.geometry.core.math.vector.UnitVector3D
 Normal to the surface.

#### Returns

### UnitVector3D

Normal unit vector to the cone at this evaluation.

property ConeEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D First derivative with respect to the U parameter.

### **Returns**

### Vector3D

First derivative with respect to the U parameter.

property ConeEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the V parameter.

### Returns

### Vector3D

First derivative with respect to the V parameter.

property ConeEvaluation.uu\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to the U parameter.

## Returns

### Vector3D

Second derivative with respect to the U parameter.

### Returns

### Vector3D

Second derivative with respect to U and V parameters.

property ConeEvaluation.vv\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to the V parameter.

### Returns

### Vector3D

Second derivative with respect to the V parameter.

 ${\tt property} \ \ {\tt ConeEvaluation.min\_curvature:} \ \ {\tt ansys.geometry.core.typing.Real}$ 

Minimum curvature of the cone.

#### Returns

Real

Minimum curvature of the cone.

property ConeEvaluation.min\_curvature\_direction:
 ansys.geometry.core.math.vector.UnitVector3D

Minimum curvature direction.

### Returns

UnitVector3D

Minimum curvature direction.

property ConeEvaluation.max\_curvature: ansys.geometry.core.typing.Real

Maximum curvature of the cone.

#### Returns

Real

Maximum curvature of the cone.

property ConeEvaluation.max\_curvature\_direction: ansys.geometry.core.math.vector.UnitVector3D

Maximum curvature direction.

## Returns

UnitVector3D

Maximum curvature direction.

# **Description**

Provides for creating and managing a cone.

The cylinder.py module

## **Summary**

#### Classes

Cylinder	Provides 3D cylinder representation.
CylinderEvaluation	Provides evaluation of a cylinder at given parameters.

## Cylinder

**class** ansys.geometry.core.shapes.surfaces.cylinder.**Cylinder**(origin: numpy.ndarray | an-

sys.geometry.core.typing.RealSequence | an-

sys.geometry.core.math.point.Point3D,

radius: pint.Quantity | an-

sys.geometry.core.misc.measurements. Distance

| ansys.geometry.core.typing.Real, reference: numpy.ndarray | an-

sys. geometry. core. typing. Real Sequence

an-

sys.geometry.core.math.vector.UnitVector3D

an-

sys.geometry.core.math.vector.Vector3D

 $= UNITVECTOR3D\_X$ , axis:

numpy.ndarray | an-

sys. geometry. core. typing. Real Sequence

an-

sys.geometry.core.math.vector.Unit Vector 3D

an-

sys.geometry.core.math.vector.Vector3D

 $= UNITVECTOR3D_Z$ )

 $Bases: \ ansys. geometry. core. shapes. surfaces. surface. Surface$ 

Provides 3D cylinder representation.

## **Parameters**

#### origin

[ndarray | RealSequence | Point3D] Origin of the cylinder.

## radius

[Quantity | Distance | Real] Radius of the cylinder.

#### reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## axis

[ndarray | RealSequence | UnitVector3D | Vector3D] Z-axis direction.

## **Overview**

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the surface.
contains_point	Check a point is contained by the surface.

#### **Methods**

surface_area	Get the surface area of the cylinder.
volume	Get the volume of the cylinder.
transformed_copy	Create a transformed copy of the cylinder from a transformation matrix.
mirrored_copy	Create a mirrored copy of the cylinder along the y-axis.
evaluate	Evaluate the cylinder at the given parameters.
<pre>project_point</pre>	Project a point onto the cylinder and evaluate the cylinder.
parameterization	Parameterize the cylinder surface as a tuple (U and V respectively).

## **Properties**

origin	Origin of the cylinder.
radius	Radius of the cylinder.
dir_x	X-direction of the cylinder.
dir_y	Y-direction of the cylinder.
dir_z	Z-direction of the cylinder.

## **Special methods**

\_\_eq\_\_ Equals operator for the Cylinder class.

## Import detail

from ansys.geometry.core.shapes.surfaces.cylinder import Cylinder

## **Property detail**

```
property Cylinder.origin: ansys.geometry.core.math.point.Point3D
    Origin of the cylinder.
```

property Cylinder.radius: pint.Quantity

Radius of the cylinder.

property Cylinder.dir\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-direction of the cylinder.

property Cylinder.dir\_y: ansys.geometry.core.math.vector.UnitVector3D
 Y-direction of the cylinder.

property Cylinder.dir\_z: ansys.geometry.core.math.vector.UnitVector3D
Z-direction of the cylinder.

#### Method detail

Cylinder.surface\_area(height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real)  $\rightarrow$  pint.Quantity

Get the surface area of the cylinder.

# **Parameters**

#### height

[Quantity | Distance | Real] Height to bound the cylinder at.

## Returns

## Quantity

Surface area of the temporarily bounded cylinder.

## **Notes**

By nature, a cylinder is infinite. If you want to get the surface area, you must bound it by a height. Normally a cylinder surface is not closed (does not have "caps" on the ends). This method assumes that the cylinder is closed for the purpose of getting the surface area.

Cylinder.volume(height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real) → pint.Quantity

Get the volume of the cylinder.

#### **Parameters**

### height

[Quantity | Distance | Real] Height to bound the cylinder at.

#### Returns

#### Quantity

Volume of the temporarily bounded cylinder.

#### **Notes**

By nature, a cylinder is infinite. If you want to get the surface area, you must bound it by a height. Normally a cylinder surface is not closed (does not have "caps" on the ends). This method assumes that the cylinder is closed for the purpose of getting the surface area.

Cylinder.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow$  Cylinder

Create a transformed copy of the cylinder from a transformation matrix.

#### **Parameters**

#### matrix

[Matrix44] 4X4 transformation matrix to apply to the cylinder.

## Returns

### Cylinder

New cylinder that is the transformed copy of the original cylinder.

Cylinder.mirrored\_copy()  $\rightarrow$  Cylinder

Create a mirrored copy of the cylinder along the y-axis.

## Returns

## Cylinder

New cylinder that is a mirrored copy of the original cylinder.

Cylinder.\_\_eq\_\_(other: Cylinder)  $\rightarrow$  bool

Equals operator for the Cylinder class.

Cylinder.evaluate(parameter: ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  CylinderEvaluation Evaluate the cylinder at the given parameters.

### **Parameters**

#### parameter

[ParamUV] Parameters (u,v) to evaluate the cylinder at.

### **Returns**

## CylinderEvaluation

Resulting evaluation.

Cylinder.project\_point(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow$  CylinderEvaluation Project a point onto the cylinder and evaluate the cylinder.

#### **Parameters**

#### point

[Point3D] Point to project onto the cylinder.

## Returns

## CylinderEvaluation

Resulting evaluation.

 $\label{continuous} {\tt Cylinder.parameterization}() \rightarrow {\tt tuple} [ansys.geometry.core.shapes.parameterization.Parameterization, ansys.geometry.core.shapes.parameterization.Parameterization]$ 

Parameterize the cylinder surface as a tuple (U and V respectively).

The U parameter specifies the clockwise angle around the axis (right-hand corkscrew law), with a zero parameter at dir\_x and a period of 2\*pi.

The V parameter specifies the distance along the axis, with a zero parameter at the XY plane of the cylinder.

#### Returns

## tuple[Parameterization, Parameterization]

Information about how a cylinder's u and v parameters are parameterized, respectively.

## abstractmethod Cylinder.contains\_param(param\_uv:

ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  bool

Check a parameter is within the parametric range of the surface.

abstractmethod Cylinder.contains\_point(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow$  bool

Check a point is contained by the surface.

The point can either lie within the surface or on its boundary.

## CylinderEvaluation

class ansys.geometry.core.shapes.surfaces.cylinder.CylinderEvaluation(cylinder: Cylinder,

parameter: an-

sys.geometry.core.shapes.parameterization.

 $Bases: \ ansys. geometry. core. shapes. surfaces. surface\_evaluation. Surface Evaluation \\$ 

Provides evaluation of a cylinder at given parameters.

#### **Parameters**

cylinder: ~ansys.geometry.core.shapes.surfaces.cylinder.Cylinder

Cylinder to evaluate.

# parameter: ParamUV

Parameters (u, v) to evaluate the cylinder at.

## **Overview**

## **Properties**

cylinder	Cylinder being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
normal	Normal to the surface.
u_derivative	First derivative with respect to the U parameter.
v_derivative	First derivative with respect to the V parameter.
uu_derivative	Second derivative with respect to the U parameter.
uv_derivative	Second derivative with respect to the U and V parameters.
vv_derivative	Second derivative with respect to the V parameter.
min_curvature	Minimum curvature of the cylinder.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature of the cylinder.
max_curvature_direction	Maximum curvature direction.

## Import detail

from ansys.geometry.core.shapes.surfaces.cylinder import CylinderEvaluation

# **Property detail**

property CylinderEvaluation.cylinder: Cylinder

Cylinder being evaluated.

property CylinderEvaluation.parameter:

ansys.geometry.core.shapes.parameterization.ParamUV

Parameter that the evaluation is based upon.

property CylinderEvaluation.position: ansys.geometry.core.math.point.Point3D

Position of the evaluation.

## Returns

#### Point3D

Point that lies on the cylinder at this evaluation.

#### Returns

#### UnitVector3D

Normal unit vector to the cylinder at this evaluation.

property CylinderEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the U parameter.

## Returns

#### Vector3D

First derivative with respect to the U parameter.

property CylinderEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the V parameter.

# Returns

### Vector3D

First derivative with respect to the V parameter.

property CylinderEvaluation.uu\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to the U parameter.

#### Returns

#### Vector3D

Second derivative with respect to the U parameter.

property CylinderEvaluation.uv\_derivative: ansys.geometry.core.math.vector.Vector3D Second derivative with respect to the U and V parameters.

#### Returns

#### Vector3D

Second derivative with respect to the U and v parameters.

property CylinderEvaluation.vv\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to the V parameter.

#### Returns

#### Vector3D

Second derivative with respect to the V parameter.

property CylinderEvaluation.min\_curvature: ansys.geometry.core.typing.Real
 Minimum curvature of the cylinder.

#### Returns

## Real

Minimum curvature of the cylinder.

property CylinderEvaluation.min\_curvature\_direction: ansys.geometry.core.math.vector.UnitVector3D

Minimum curvature direction.

## Returns

## UnitVector3D

Mminimum curvature direction.

#### Returns

#### Real

Maximum curvature of the cylinder.

property CylinderEvaluation.max\_curvature\_direction: ansys.geometry.core.math.vector.UnitVector3D

Maximum curvature direction.

### Returns

#### UnitVector3D

Maximum curvature direction.

# **Description**

Provides for creating and managing a cylinder.

## The plane.py module

# **Summary**

## **Classes**

PlaneSurface	Provides 3D plane surface representation.
PlaneEvaluation	Provides evaluation of a plane at given parameters.

## PlaneSurface

class ansys.geometry.core.shapes.surfaces.plane.PlaneSurface(origin: numpy.ndarray | an-

sys.geometry.core.typing.RealSequence | ansys.geometry.core.math.point.Point3D, reference: numpy.ndarray | ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_X, axis: numpy.ndarray | ansys.geometry.core.typing.RealSequence | ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D

= UNITVECTOR3D Z

Bases: ansys.geometry.core.shapes.surfaces.surface.Surface

Provides 3D plane surface representation.

# **Parameters**

#### origin

[ndarray | RealSequence | Point 3D] Centered origin of the plane.

#### reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## axis

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## **Overview**

## **Abstract methods**

contains_param	Check a ParamUV is within the parametric range of the surface.
contains_point	Check whether a 3D point is in the domain of the plane.

## **Methods**

parameterization	Parametrize the plane.
<pre>project_point</pre>	Evaluate the plane at a given 3D point.
transformed_copy	Get a transformed version of the plane given the transform.
evaluate	Evaluate the plane at a given u and v parameter.

## **Properties**

origin	Origin of the plane.
dir_x	X-direction of the plane.
dir_y	Y-direction of the plane.
dir_z	Z-direction of the plane.

# **Special methods**

**\_\_eq\_\_** Check whether two planes are equal.

## Import detail

from ansys.geometry.core.shapes.surfaces.plane import PlaneSurface

## **Property detail**

property PlaneSurface.origin: ansys.geometry.core.math.point.Point3D
 Origin of the plane.

property PlaneSurface.dir\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-direction of the plane.

property PlaneSurface.dir\_y: ansys.geometry.core.math.vector.UnitVector3D
 Y-direction of the plane.

property PlaneSurface.dir\_z: ansys.geometry.core.math.vector.UnitVector3D
 Z-direction of the plane.

## **Method detail**

PlaneSurface.\_\_eq\_\_(other: PlaneSurface)  $\rightarrow$  bool Check whether two planes are equal.

abstractmethod PlaneSurface.contains\_param(param\_uv:

ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  bool

Check a ParamUV is within the parametric range of the surface.

 $\textbf{abstractmethod} \ \ PlaneSurface. \textbf{contains\_point}(point: \ ansys.geometry.core.math.point.Point3D) \rightarrow bool \\ Check \ whether \ a \ 3D \ point \ is \ in \ the \ domain \ of \ the \ plane.$ 

PlaneSurface.parameterization()  $\rightarrow$  tuple[ansys.geometry.core.shapes.parameterization.Parameterization, ansys.geometry.core.shapes.parameterization.Parameterization]

Parametrize the plane.

PlaneSurface.project\_point(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow$ 

 $ansys. geometry. core. shapes. surfaces. surface\_evaluation. Surface Evaluation$ 

Evaluate the plane at a given 3D point.

PlaneSurface.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow$  ansys.geometry.core.shapes.surfaces.surfaces.Surface

Get a transformed version of the plane given the transform.

 $\label{eq:parameter:ansys} Plane Surface. \textbf{evaluate}(parameter: ansys. geometry. core. shapes. parameterization. ParamUV) \rightarrow Plane Evaluation$ 

Evaluate the plane at a given u and v parameter.

## PlaneEvaluation

class ansys.geometry.core.shapes.surfaces.plane.PlaneEvaluation(plane: PlaneSurface, parameter:

an-

sys.geometry.core.shapes.parameterization.ParamU

Bases: ansys.geometry.core.shapes.surfaces.surface\_evaluation.SurfaceEvaluation

Provides evaluation of a plane at given parameters.

#### **Parameters**

plane: ~ansys.geometry.core.shapes.surfaces.plane.PlaneSurface

Plane to evaluate.

parameter: ParamUV

Parameters (u, v) to evaluate the plane at.

#### Overview

# **Properties**

plane	Plane being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Point on the surface, based on the evaluation.
normal	Normal to the surface.
u_derivative	First derivative with respect to u.
v_derivative	First derivative with respect to v.
uu_derivative	Second derivative with respect to u.
uv_derivative	Second derivative with respect to u and v.
vv_derivative	Second derivative with respect to v.
min_curvature	Minimum curvature.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature.
max_curvature_direction	Maximum curvature direction.

## Import detail

from ansys.geometry.core.shapes.surfaces.plane import PlaneEvaluation

## **Property detail**

property PlaneEvaluation.plane: PlaneSurface
 Plane being evaluated.

property PlaneEvaluation.parameter: ansys.geometry.core.shapes.parameterization.ParamUV
Parameter that the evaluation is based upon.

property PlaneEvaluation.position: ansys.geometry.core.math.point.Point3D
Point on the surface. based on the evaluation.

property PlaneEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to u.

property PlaneEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to v.

property PlaneEvaluation.vv\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to v.

property PlaneEvaluation.min\_curvature\_direction:
 ansys.geometry.core.math.vector.UnitVector3D

Minimum curvature direction.

property PlaneEvaluation.max\_curvature\_direction:
ansys.geometry.core.math.vector.UnitVector3D

Maximum curvature direction.

#### **Description**

Provides for creating and managing a plane.

The sphere.py module

# **Summary**

## Classes

Sphere	Provides 3D sphere representation.
SphereEvaluation	Evaluate a sphere at given parameters.

## Sphere

```
class ansys.geometry.core.shapes.surfaces.sphere.Sphere(origin: numpy.ndarray |
                                                                   ansys.geometry.core.typing.RealSequence |
                                                                   ansys.geometry.core.math.point.Point3D,
                                                                   radius: pint.Quantity | an-
                                                                   sys.geometry.core.misc.measurements. Distance\\
                                                                   | ansys.geometry.core.typing.Real,
                                                                   reference: numpy.ndarray |
                                                                   ansys.geometry.core.typing.RealSequence |
                                                                   sys.geometry.core.math.vector.UnitVector3D
                                                                   ansys.geometry.core.math.vector.Vector3D
                                                                   = UNITVECTOR3D_X, axis:
                                                                   numpy.ndarray
                                                                   ansys.geometry.core.typing.RealSequence |
                                                                   sys.geometry.core.math.vector.UnitVector3D
                                                                   ansys.geometry.core.math.vector.Vector3D
                                                                   = UNITVECTOR3D_Z
```

Bases: ansys.geometry.core.shapes.surfaces.surface.Surface

Provides 3D sphere representation.

## **Parameters**

## origin

[ndarray | RealSequence | Point3D] Origin of the sphere.

#### radius

[Quantity | Distance | Real] Radius of the sphere.

## reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

## axis

[ndarray | RealSequence | UnitVector3D | Vector3D] Z-axis direction.

## **Overview**

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the surface.
contains_point	Check a point is contained by the surface.

## Methods

transformed_copy	Create a transformed copy of the sphere from a transformation matrix.
mirrored_copy	Create a mirrored copy of the sphere along the y-axis.
evaluate	Evaluate the sphere at the given parameters.
<pre>project_point</pre>	Project a point onto the sphere and evaluate the sphere.
parameterization	Parameterization of the sphere surface as a tuple (U, V).

# **Properties**

origin	Origin of the sphere.
radius	Radius of the sphere.
dir_x	X-direction of the sphere.
dir_y	Y-direction of the sphere.
dir_z	Z-direction of the sphere.
surface_area	Surface area of the sphere.
volume	Volume of the sphere.

# **Special methods**

eq Equals operator for the Sphere	class.
-----------------------------------	--------

# Import detail

```
from ansys.geometry.core.shapes.surfaces.sphere import Sphere
```

## **Property detail**

```
property Sphere.origin: ansys.geometry.core.math.point.Point3D
    Origin of the sphere.
```

property Sphere.radius: pint.Quantity

Radius of the sphere.

property Sphere.dir\_x: ansys.geometry.core.math.vector.UnitVector3D
 X-direction of the sphere.

property Sphere.dir\_y: ansys.geometry.core.math.vector.UnitVector3D
 Y-direction of the sphere.

property Sphere.dir\_z: ansys.geometry.core.math.vector.UnitVector3D
 Z-direction of the sphere.

property Sphere.surface\_area: pint.Quantity

Surface area of the sphere.

property Sphere.volume: pint.Quantity

Volume of the sphere.

## **Method detail**

Sphere.**\_\_eq**\_\_(other: Sphere)  $\rightarrow$  bool

Equals operator for the Sphere class.

Sphere.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow$  Sphere

Create a transformed copy of the sphere from a transformation matrix.

#### **Parameters**

#### matrix

[Matrix44] 4X4 transformation matrix to apply to the sphere.

#### Returns

#### Sphere

New sphere that is the transformed copy of the original sphere.

 $\texttt{Sphere.mirrored\_copy()} \rightarrow \textit{Sphere}$ 

Create a mirrored copy of the sphere along the y-axis.

#### Returns

### Sphere

New sphere that is a mirrored copy of the original sphere.

Sphere.evaluate(parameter: ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  SphereEvaluation Evaluate the sphere at the given parameters.

#### **Parameters**

#### parameter

[ParamUV] Parameters (u,v) to evaluate the sphere at.

## Returns

# ${\bf Sphere Evaluation}$

Resulting evaluation.

 $Sphere.\textbf{project\_point}(\textit{point}: ansys.geometry.core.math.point.Point3D) \rightarrow \textit{SphereEvaluation}$ 

Project a point onto the sphere and evaluate the sphere.

#### **Parameters**

### point

[Point3D] Point to project onto the sphere.

#### Returns

#### **SphereEvaluation**

Resulting evaluation.

Sphere.parameterization()  $\rightarrow$  tuple[ansys.geometry.core.shapes.parameterization.Parameterization, ansys.geometry.core.shapes.parameterization.Parameterization]

Parameterization of the sphere surface as a tuple (U, V).

The U parameter specifies the longitude angle, increasing clockwise (east) about dir\_z (right-hand corkscrew law). It has a zero parameter at dir\_x and a period of 2\*pi.

The V parameter specifies the latitude, increasing north, with a zero parameter at the equator and a range of [-pi/2, pi/2].

## Returns

## tuple[Parameterization, Parameterization]

Information about how a sphere's u and v parameters are parameterized, respectively.

**abstractmethod** Sphere.contains\_param(param\_uv: ansys.geometry.core.shapes.parameterization.ParamUV)

→ bool

Check a parameter is within the parametric range of the surface.

**abstractmethod** Sphere.contains\_point(point: ansys.geometry.core.math.point.Point3D) → bool

Check a point is contained by the surface.

The point can either lie within the surface or on its boundary.

## **SphereEvaluation**

**class** ansys.geometry.core.shapes.surfaces.sphere.**SphereEvaluation**(*sphere*: Sphere, *parameter*:

ın-

sys.geometry.core.shapes.parameterization.Param

 $Bases: \ ansys.geometry.core.shapes.surfaces.surface\_evaluation.SurfaceEvaluation$ 

Evaluate a sphere at given parameters.

#### **Parameters**

sphere: ~ansys.geometry.core.shapes.surfaces.sphere.Sphere

Sphere to evaluate.

parameter: ParamUV

Parameters (u, v) to evaluate the sphere at.

## Overview

# **Properties**

sphere	Sphere being evaluated.
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
normal	The normal to the surface.
u_derivative	First derivative with respect to the U parameter.
v_derivative	First derivative with respect to the V parameter.
uu_derivative	Second derivative with respect to the U parameter.
uv_derivative	Second derivative with respect to the U and V parameters.
vv_derivative	Second derivative with respect to the V parameter.
min_curvature	Minimum curvature of the sphere.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature of the sphere.
<pre>max_curvature_direction</pre>	Maximum curvature direction.

# Import detail

from ansys.geometry.core.shapes.surfaces.sphere import SphereEvaluation

## **Property detail**

property SphereEvaluation.sphere: Sphere

Sphere being evaluated.

property SphereEvaluation.parameter: ansys.geometry.core.shapes.parameterization.ParamUV Parameter that the evaluation is based upon.

property SphereEvaluation.position: ansys.geometry.core.math.point.Point3D
 Position of the evaluation.

#### Returns

#### Point3D

Point that lies on the sphere at this evaluation.

property SphereEvaluation.normal: ansys.geometry.core.math.vector.UnitVector3D
The normal to the surface.

#### Returns

#### UnitVector3D

Normal unit vector to the sphere at this evaluation.

property SphereEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D First derivative with respect to the U parameter.

#### **Returns**

#### Vector3D

First derivative with respect to the U parameter.

property SphereEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the V parameter.

#### Returns

#### Vector3D

First derivative with respect to the V parameter.

### Returns

#### Vector3D

Second derivative with respect to the U parameter.

### Returns

## Vector3D

The second derivative with respect to the U and V parameters.

#### Returns

#### Vector3D

The second derivative with respect to the V parameter.

### Returns

Real

Minimum curvature of the sphere.

property SphereEvaluation.min\_curvature\_direction:
 ansys.geometry.core.math.vector.UnitVector3D

Minimum curvature direction.

### Returns

UnitVector3D

Minimum curvature direction.

## Returns

Real

Maximum curvature of the sphere.

property SphereEvaluation.max\_curvature\_direction:
 ansys.geometry.core.math.vector.UnitVector3D

Maximum curvature direction.

#### **Returns**

UnitVector3D

Maximum curvature direction.

## **Description**

Provides for creating and managing a sphere.

The surface.py module

**Summary** 

Classes

Surface Provides the abstract base class for a 3D surface.

#### Surface

class ansys.geometry.core.shapes.surfaces.surface.

Bases: abc.ABC

Provides the abstract base class for a 3D surface.

## **Overview**

## **Abstract methods**

parameterization	Parameterize the surface as a tuple (U and V respectively).
contains_param	Check a parameter is within the parametric range of the surface.
contains_point	Check a point is contained by the surface.
transformed_copy	Create a transformed copy of the surface.
eq	Determine if two surfaces are equal.
evaluate	Evaluate the surface at the given parameter.
<pre>project_point</pre>	Project a point to the surface.

#### **Methods**

## Import detail

from ansys.geometry.core.shapes.surfaces.surface import Surface

#### Method detail

abstractmethod Surface.parameterization()  $\rightarrow tu$ -

ple[ansys.geometry.core.shapes.parameterization.Parameterization, ansys.geometry.core.shapes.parameterization.Parameterization]

Parameterize the surface as a tuple (U and V respectively).

abstractmethod Surface.contains\_param(param\_uv:

ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  bool

Check a parameter is within the parametric range of the surface.

 $\textbf{abstractmethod} \ \, \textbf{Surface.contains\_point} (\textit{point:} \ \, \textbf{ansys.geometry.core.math.point.Point3D)} \rightarrow \textbf{bool}$ 

Check a point is contained by the surface.

The point can either lie within the surface or on its boundary.

**abstractmethod** Surface.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow$  Surface Create a transformed copy of the surface.

**abstractmethod** Surface.**\_\_eq\_\_**(other: Surface)  $\rightarrow$  bool

Determine if two surfaces are equal.

 $\textbf{abstractmethod} \ \ \textbf{Surface.evaluate} (\textit{parameter:} \ ansys.geometry.core.shapes.parameterization.ParamUV) \rightarrow \textbf{Approximate Parameter})$ 

an

 $sys. geometry. core. shapes. surfaces. surface\_evaluation. Surface Evaluation$ 

Evaluate the surface at the given parameter.

**abstractmethod** Surface.**project\_point**(point: ansys.geometry.core.math.point.Point3D)  $\rightarrow an$ -

 $sys.geometry.core.shapes.surfaces.surface\_evaluation.SurfaceEvaluation$ 

Project a point to the surface.

This method returns the evaluation at the closest point.

 $Surface. \textbf{trim}(box\_uv: ansys.geometry.core.shapes.box\_uv.BoxUV) \rightarrow ansys.geometry.core.shapes.surfaces.trimmed\_surface.TrimmedSurface$ 

Trim this surface by bounding it with a BoxUV.

#### **Returns**

## TrimmedSurface

The resulting bounded surface.

# **Description**

Provides the Surface class.

# The surface\_evaluation.py module

# **Summary**

#### Classes

## SurfaceEvaluation

 ${\bf class} \ {\bf ansys.geometry.core.shapes.surfaces.surface\_evaluation.} {\bf SurfaceEvaluation} ({\it parameter:} {\bf class}) {\bf class} \ {\bf class}) {\bf class} {\bf class}$ 

an-

sys.geometry.core.shapes.param

Provides for evaluating a surface.

#### Overview

## **Properties**

parameter	Parameter that the evaluation is based upon.
position	Point on the surface, based on the evaluation.
normal	Normal to the surface.
u_derivative	First derivative with respect to the U parameter.
v_derivative	First derivative with respect to the V parameter.
uu_derivative	Second derivative with respect to the U parameter.
uv_derivative	The second derivative with respect to the U and V parameters.
vv_derivative	The second derivative with respect to v.
min_curvature	Minimum curvature.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature.
<pre>max_curvature_direction</pre>	Maximum curvature direction.

# Import detail

from ansys.geometry.core.shapes.surfaces.surface\_evaluation import SurfaceEvaluation

## **Property detail**

property SurfaceEvaluation.parameter: ansys.geometry.core.shapes.parameterization.ParamUV

#### Abstractmethod

Parameter that the evaluation is based upon.

property SurfaceEvaluation.position: ansys.geometry.core.math.point.Point3D

#### Abstractmethod

Point on the surface, based on the evaluation.

property SurfaceEvaluation.normal: ansys.geometry.core.math.vector.UnitVector3D

#### Abstractmethod

Normal to the surface.

property SurfaceEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D

#### Abstractmethod

First derivative with respect to the U parameter.

property SurfaceEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D

#### Abstractmethod

First derivative with respect to the V parameter.

property SurfaceEvaluation.uu\_derivative: ansys.geometry.core.math.vector.Vector3D

#### Abstractmethod

Second derivative with respect to the U parameter.

property SurfaceEvaluation.uv\_derivative: ansys.geometry.core.math.vector.Vector3D

#### Abstractmethod

The second derivative with respect to the U and V parameters.

property SurfaceEvaluation.vv\_derivative: ansys.geometry.core.math.vector.Vector3D

### Abstractmethod

The second derivative with respect to v.

property SurfaceEvaluation.min\_curvature: ansys.geometry.core.typing.Real

#### Abstractmethod

Minimum curvature.

property SurfaceEvaluation.min\_curvature\_direction: ansys.geometry.core.math.vector.UnitVector3D

# Abstractmethod

Minimum curvature direction.

property SurfaceEvaluation.max\_curvature: ansys.geometry.core.typing.Real

#### Abstractmethod

Maximum curvature.

property SurfaceEvaluation.max\_curvature\_direction:
 ansys.geometry.core.math.vector.UnitVector3D

#### Abstractmethod

Maximum curvature direction.

## **Description**

Provides for evaluating a surface.

## The torus.py module

## **Summary**

#### Classes

Torus	Provides 3D torus representation.
TorusEvaluation	Evaluate the torus`` at given parameters.

### **Torus**

class ansys.geometry.core.shapes.surfaces.torus.Torus(origin: numpy.ndarray |

ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.point.Point3D, major\_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, minor\_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, reference: numpy.ndarray ansys.geometry.core.typing.RealSequence | sys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D\_X, axis: numpy.ndarray ansys.geometry.core.typing.RealSequence | ansys.geometry.core.math.vector.UnitVector3D | ansys.geometry.core.math.vector.Vector3D = UNITVECTOR3D Z)

Bases: ansys.geometry.core.shapes.surfaces.surface.Surface

Provides 3D torus representation.

## **Parameters**

## origin

[ndarray | RealSequence | Point3D] Centered origin of the torus.

## major\_radius

[Quantity | Distance | Real] Major radius of the torus.

## minor\_radius

[Quantity | Distance | Real] Minor radius of the torus.

## reference

[ndarray | RealSequence | UnitVector3D | Vector3D] X-axis direction.

#### axis

 $[ndarray \mid Real Sequence \mid Unit Vector 3D \mid Vector 3D] \ Z\ -axis \ direction.$ 

## Overview

## **Abstract methods**

contains_param	Check a parameter is within the parametric range of the surface.
contains_point	Check a point is contained by the surface.

## **Methods**

transformed_copy	Create a transformed copy of the torus from a transformation matrix.
mirrored_copy	Create a mirrored copy of the torus along the y-axis.
evaluate	Evaluate the torus at the given parameters.
parameterization	Parameterize the torus surface as a tuple (U and V respectively).
<pre>project_point</pre>	Project a point onto the torus and evaluate the torus.

# **Properties**

origin	Origin of the torus.
major_radius	Semi-major radius of the torus.
minor_radius	Semi-minor radius of the torus.
dir_x	X-direction of the torus.
dir_y	Y-direction of the torus.
dir_z	Z-direction of the torus.
volume	Volume of the torus.
surface_area	Surface_area of the torus.

# **Special methods**

\_\_eq\_\_ Equals operator for the Torus class.

# Import detail

from ansys.geometry.core.shapes.surfaces.torus import Torus

# **Property detail**

property Torus.origin: ansys.geometry.core.math.point.Point3D

Origin of the torus.

property Torus.major\_radius: pint.Quantity

Semi-major radius of the torus.

property Torus.minor\_radius: pint.Quantity

Semi-minor radius of the torus.

property Torus.dir\_x: ansys.geometry.core.math.vector.UnitVector3D

X-direction of the torus.

property Torus.dir\_y: ansys.geometry.core.math.vector.UnitVector3D

Y-direction of the torus.

property Torus.dir\_z: ansys.geometry.core.math.vector.UnitVector3D

Z-direction of the torus.

property Torus.volume: pint.Quantity

Volume of the torus.

property Torus.surface\_area: pint.Quantity

Surface\_area of the torus.

## **Method detail**

Torus. $_{-}$ eq $_{-}$ (other: Torus)  $\rightarrow$  bool

Equals operator for the Torus class.

Torus.transformed\_copy(matrix: ansys.geometry.core.math.matrix.Matrix44)  $\rightarrow Torus$ 

Create a transformed copy of the torus from a transformation matrix.

#### **Parameters**

matrix

[Matrix44] 4x4 transformation matrix to apply to the torus.

## Returns

Torus

New torus that is the transformed copy of the original torus.

 $\texttt{Torus.mirrored\_copy()} \to \textit{Torus}$ 

Create a mirrored copy of the torus along the y-axis.

### **Returns**

Torus

New torus that is a mirrored copy of the original torus.

Torus.evaluate(parameter: ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  TorusEvaluation Evaluate the torus at the given parameters.

### **Parameters**

parameter

[ParamUV] Parameters (u,v) to evaluate the torus at.

#### Returns

#### **TorusEvaluation**

Resulting evaluation.

 $\label{torus.parameterization} Torus. \textbf{parameterization}() \rightarrow tuple [\textit{ansys.geometry.core.shapes.parameterization.Parameterization}, \\ \textit{ansys.geometry.core.shapes.parameterization.Parameterization}]$ 

Parameterize the torus surface as a tuple (U and V respectively).

The U parameter specifies the longitude angle, increasing clockwise (east) about the axis (right-hand corkscrew law). It has a zero parameter at Geometry.Frame.DirX and a period of 2\*pi.

The V parameter specifies the latitude, increasing north, with a zero parameter at the equator. For the donut, where the major radius is greater than the minor radius, the range is [-pi, pi] and the parameterization is periodic. For a degenerate torus, the range is restricted accordingly and the parameterization is non-periodic.

#### **Returns**

## tuple[Parameterization, Parameterization]

Information about how a torus's u and v parameters are parameterized, respectively.

 $\textbf{Torus.project\_point}(\textit{point:} \ ansys.geometry.core.math.point.Point3D) \rightarrow \textit{TorusEvaluation}$ 

Project a point onto the torus and evaluate the torus.

#### **Parameters**

### point

[Point3D] Point to project onto the torus.

#### Returns

#### **TorusEvaluation**

Resulting evaluation.

 $\begin{tabular}{ll} \textbf{abstractmethod} & \textbf{Torus.contains\_param}(param\_uv: ansys.geometry.core.shapes.parameterization.ParamUV) \\ &\rightarrow bool \end{tabular}$ 

Check a parameter is within the parametric range of the surface.

 $\textbf{abstractmethod} \ \ \textbf{Torus.contains\_point}(\textit{point:} \ ansys.geometry.core.math.point.Point3D) \rightarrow bool$ 

Check a point is contained by the surface.

The point can either lie within the surface or on its boundary.

## **TorusEvaluation**

class ansys.geometry.core.shapes.surfaces.torus.TorusEvaluation(torus: Torus, parameter: an-

sys.geometry.core.shapes.parameterization.ParamU

Bases: ansys.geometry.core.shapes.surfaces.surface\_evaluation.SurfaceEvaluation

Evaluate the torus`` at given parameters.

#### **Parameters**

Torus: ~ansys.geometry.core.shapes.surfaces.torus.Torus

Torust to evaluate.

### parameter: ParamUV

Parameters (u, v) to evaluate the torus at.

## **Overview**

## **Properties**

torus	Torus being evaluated.
torus	
parameter	Parameter that the evaluation is based upon.
position	Position of the evaluation.
normal	Normal to the surface.
u_derivative	First derivative with respect to the U parameter.
v_derivative	First derivative with respect to the V parameter.
uu_derivative	Second derivative with respect to the U parameter.
uv_derivative	Second derivative with respect to the U and V parameters.
vv_derivative	Second derivative with respect to the V parameter.
curvature	Curvature of the torus.
min_curvature	Minimum curvature of the torus.
<pre>min_curvature_direction</pre>	Minimum curvature direction.
max_curvature	Maximum curvature of the torus.
<pre>max_curvature_direction</pre>	Maximum curvature direction.

## **Attributes**

cache

# Import detail

from ansys.geometry.core.shapes.surfaces.torus import TorusEvaluation

# **Property detail**

property TorusEvaluation.torus: Torus

Torus being evaluated.

property TorusEvaluation.parameter: ansys.geometry.core.shapes.parameterization.ParamUV
 Parameter that the evaluation is based upon.

property TorusEvaluation.position: ansys.geometry.core.math.point.Point3D
 Position of the evaluation.

## Returns

### Point3D

Point that lies on the torus at this evaluation.

property TorusEvaluation.normal: ansys.geometry.core.math.vector.UnitVector3D
 Normal to the surface.

## Returns

## UnitVector3D

Normal unit vector to the torus at this evaluation.

property TorusEvaluation.u\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the U parameter.

### Returns

274

#### Vector3D

First derivative with respect to the U parameter.

property TorusEvaluation.v\_derivative: ansys.geometry.core.math.vector.Vector3D
First derivative with respect to the V parameter.

#### Returns

#### Vector3D

First derivative with respect to the V parameter.

property TorusEvaluation.uu\_derivative: ansys.geometry.core.math.vector.Vector3D Second derivative with respect to the U parameter.

#### Returns

#### Vector3D

Second derivative with respect to the U parameter.

property TorusEvaluation.uv\_derivative: ansys.geometry.core.math.vector.Vector3D Second derivative with respect to the U and V parameters.

#### Returns

#### Vector3D

Second derivative with respect to the U and V parameters.

property TorusEvaluation.vv\_derivative: ansys.geometry.core.math.vector.Vector3D
Second derivative with respect to the V parameter.

#### Returns

## Vector3D

Second derivative with respect to the V parameter.

property TorusEvaluation.curvature: tuple[ansys.geometry.core.typing.Real, ansys.geometry.core.math.vector.Vector3D, ansys.geometry.core.typing.Real, ansys.geometry.core.math.vector.Vector3D]

Curvature of the torus.

### Returns

```
tuple[Real, Vector3D, Real, Vector3D]
```

Minimum and maximum curvature value and direction, respectively.

#### **Returns**

## Real

Minimum curvature of the torus.

property TorusEvaluation.min\_curvature\_direction:
ansys.geometry.core.math.vector.UnitVector3D

Minimum curvature direction.

#### Returns

#### UnitVector3D

Minimum curvature direction.

property TorusEvaluation.max\_curvature: ansys.geometry.core.typing.Real

Maximum curvature of the torus.

#### Returns

Real

Maximum curvature of the torus.

property TorusEvaluation.max\_curvature\_direction: ansys.geometry.core.math.vector.UnitVector3D

Maximum curvature direction.

#### Returns

UnitVector3D

Maximum curvature direction.

## **Attribute detail**

TorusEvaluation.cache

## **Description**

Provides for creating and managing a torus.

The trimmed\_surface.py module

## **Summary**

#### Classes

TrimmedSurface	Represents a trimmed surface.
Reversed Trimmed Surface	Represents a reversed trimmed surface.

## TrimmedSurface

class ansys.geometry.core.shapes.surfaces.trimmed\_surface.TrimmedSurface(geometry: an-

sys.geometry.core.shapes.surfaces.surfa box\_uv: an-

sys.geometry.core.shapes.box\_uv.BoxU

Represents a trimmed surface.

A trimmed surface is a surface that has a boundary. This boundary comes in the form of a bounding BoxUV.

## **Parameters**

face

[Face] Face that the trimmed surface belongs to.

## geometry

[Surface] Underlying mathematical representation of the surface.

## **Overview**

## **Methods**

<pre>get_proportional_parameters</pre>	Convert non-proportional parameters into proportional parameters.
normal	Provide the normal to the surface.
<pre>project_point</pre>	Project a point onto the surface and evaluate it at that location.
evaluate_proportion	Evaluate the surface at proportional u and v parameters.

## **Properties**

geometry	Underlying mathematical surface.
box_uv	Bounding BoxUV of the surface.

## Import detail

from ansys.geometry.core.shapes.surfaces.trimmed\_surface import TrimmedSurface

## **Property detail**

property TrimmedSurface.geometry: ansys.geometry.core.shapes.surfaces.surface.Surface
 Underlying mathematical surface.

### **Method detail**

TrimmedSurface.get\_proportional\_parameters(param\_uv:

ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  ansys.geometry.core.shapes.parameterization.ParamUV

Convert non-proportional parameters into proportional parameters.

## **Parameters**

### param uv

[ParamUV] Non-proportional UV parameters.

#### Returns

## ParamUV

Proportional (from 0-1) UV parameters.

 $\label{thm:core.typing.Real} TrimmedSurface. \textbf{normal} (\textit{u: ansys.geometry.core.typing.Real}, \textit{v: ansys.geometry.core.typing.Real}) \rightarrow \textit{ansys.geometry.core.math.vector.UnitVector3D}$ 

Provide the normal to the surface.

#### **Parameters**

u

[Real] First coordinate of the 2D representation of a surface in UV space.

 $\mathbf{v}$ 

[Real] Second coordinate of the 2D representation of a surface in UV space.

#### Returns

#### UnitVector3D

Unit vector is normal to the surface at the given UV coordinates.

TrimmedSurface.project\_point(point: ansys.geometry.core.math.point.Point3D) →

 $ansys. geometry. core. shapes. surfaces. surface\_evaluation. Surface Evaluation$ 

Project a point onto the surface and evaluate it at that location.

#### **Parameters**

## point

[Point3D] Point to project onto the surface.

#### Returns

#### SurfaceEvaluation

Resulting evaluation.

TrimmedSurface.evaluate\_proportion(u: ansys.geometry.core.typing.Real, v:

 $ansys.geometry.core.typing.Real) \rightarrow an-$ 

 $sys. geometry. core. shapes. surface \underline{\ \ } evaluation. Surface Evaluation$ 

Evaluate the surface at proportional u and v parameters.

#### **Parameters**

u

[Real] U parameter in the proportional range [0,1].

V

[Real] V parameter in the proportional range [0,1].

### Returns

## SurfaceEvaluation

Resulting surface evaluation.

#### ReversedTrimmedSurface

 ${\bf class} \ {\bf ansys.geometry.core.shapes.surfaces.trimmed\_surface.} {\bf ReversedTrimmedSurface} ({\it geometry:} {\bf class}) {\bf class} \ {\bf ansys.geometry.core.shapes.surfaces.trimmed\_surface.} {\bf ReversedTrimmedSurface} ({\it geometry:} {\bf class}) {\bf class} \ {\bf class} \$ 

an-

sys.geometry.core.shapes.sur box\_uv:

*оол\_и* 

ansys.geometry.core.shapes.box

Bases: TrimmedSurface

Represents a reversed trimmed surface.

When a surface is reversed, its normal vector is negated to provide the proper outward facing vector.

#### **Parameters**

#### face

[Face] Face that the trimmed surface belongs to.

## geometry

[Surface] Underlying mathematical representation of the surface.

## **Overview**

## **Methods**

normal	Provide the normal to the surface.
<pre>project_point</pre>	Project a point onto the surface and evaluate it at that location.

## Import detail

from ansys.geometry.core.shapes.surfaces.trimmed\_surface import ReversedTrimmedSurface

## **Method detail**

ReversedTrimmedSurface.normal( $u: ansys.geometry.core.typing.Real, v: ansys.geometry.core.typing.Real) <math>\rightarrow ansys.geometry.core.math.vector.UnitVector3D$ 

Provide the normal to the surface.

## **Parameters**

u

[Real] First coordinate of the 2D representation of a surface in UV space.

v

[Real] Second coordinate of the 2D representation of a surface in UV space.

#### Returns

## UnitVector3D

Unit vector is normal to the surface at the given UV coordinates.

 $Reversed Trimmed Surface. \textbf{project\_point}(point: ansys. geometry. core. math. point. Point 3D) \rightarrow ansys. geometry. core. shapes. surface\_evaluation. SurfaceEvaluation$ 

Project a point onto the surface and evaluate it at that location.

### **Parameters**

## point

[Point3D] Point to project onto the surface.

### Returns

## SurfaceEvaluation

Resulting evaluation.

## **Description**

Provides the TrimmedSurface class.

## **Description**

Provides the PyAnsys Geometry surface subpackage.

# The box\_uv.py module

## **Summary**

## **Classes**

**BoxUV** Provides the implementation for BoxUV class.

## **Enums**

## **BoxUV**

class ansys.geometry.core.shapes.box\_uv.BoxUV(range\_u:

ansys.geometry.core.shapes.parameterization.Interval = None, range\_v:
ansys.geometry.core.shapes.parameterization.Interval = None)

Provides the implementation for BoxUV class.

## **Overview**

## **Methods**

is_empty	Check if this BoxUV is empty.
proportion	Evaluate the BoxUV at the given proportions.
get_center	Evaluate the this BoxUV in the center.
is_negative	Check whether the BoxUV is negative.
contains	Check whether the BoxUV contains a given u and v pair parameter.
inflate	Enlarge the BoxUV u and v intervals by deltas.
get_corner	Get the corner location of the BoxUV.

# **Properties**

interval_u	u interval.
interval_v	v interval.

# **Special methods**

eq	Check whether two BoxUV instances are equal.
ne	Check whether two BoxUV instances are not equal.

## Import detail

from ansys.geometry.core.shapes.box\_uv import BoxUV

## **Property detail**

property BoxUV.interval\_v: ansys.geometry.core.shapes.parameterization.Interval v interval.

#### **Method detail**

BoxUV. $\_$ eq $\_$ (other: object)  $\rightarrow$  bool

Check whether two BoxUV instances are equal.

BoxUV.\_\_ne\_\_(other: object)  $\rightarrow$  bool

Check whether two BoxUV instances are not equal.

BoxUV.is\_empty()

Check if this BoxUV is empty.

BoxUV.proportion(prop\_u: ansys.geometry.core.typing.Real, prop\_v: ansys.geometry.core.typing.Real)  $\rightarrow$  ansys.geometry.core.shapes.parameterization.ParamUV

Evaluate the BoxUV at the given proportions.

 $\texttt{BoxUV}.\texttt{get\_center}() o ansys.geometry.core.shapes.parameterization.ParamUV$ 

Evaluate the this BoxUV in the center.

BoxUV.is\_negative(tolerance\_u: ansys.geometry.core.typing.Real, tolerance\_v: ansys.geometry.core.typing.Real)  $\rightarrow$  bool

Check whether the BoxUV is negative.

BoxUV.contains (param: ansys.geometry.core.shapes.parameterization.ParamUV)  $\rightarrow$  bool Check whether the BoxUV contains a given u and v pair parameter.

BoxUV.inflate( $delta\_u$ : ansys.geometry.core.typing.Real,  $delta\_v$ : ansys.geometry.core.typing.Real)  $\rightarrow BoxUV$  Enlarge the BoxUV u and v intervals by deltas.

BoxUV.get\_corner(location: LocationUV)  $\rightarrow$  ansys.geometry.core.shapes.parameterization.ParamUV Get the corner location of the BoxUV.

## LocationUV

class ansys.geometry.core.shapes.box\_uv.LocationUV(\*args, \*\*kwds)

Bases: enum. Enum

Provides the enumeration for indicating locations for BoxUV.

#### Overview

## **Attributes**

TopLeft
TopCenter
TopRight
BottomLeft
BottomCenter
BottomRight
LeftCenter
RightCenter
Center

# Import detail

from ansys.geometry.core.shapes.box\_uv import LocationUV

#### Attribute detail

LocationUV.TopLeft = 1

LocationUV.TopCenter = 2

LocationUV.TopRight = 3

LocationUV.BottomLeft = 4

LocationUV.BottomCenter = 5

LocationUV.BottomRight = 6

LocationUV.LeftCenter = 7

LocationUV.RightCenter = 8

LocationUV.Center = 9

# **Description**

Provides the BoxUV class.

# The parameterization.py module

# **Summary**

## **Classes**

ParamUV	Parameter class containing 2 parameters: (u, v).
Interval	Interval class that defines a range of values.
Parameterization	Parameterization class describes the parameters of a specific geometry.

## **Enums**

ParamForm	ParamForm enum class that defines the form of a Parameterization.
ParamType	ParamType enum class that defines the type of a Parameterization.

## ParamUV

Parameter class containing 2 parameters: (u, v).

## **Parameters**

**u** [Real] U parameter.

V

[Real] V parameter.

#### **Notes**

Likened to a 2D point in UV space. Used as an argument in parametric surface evaluations. This matches the service implementation for the Geometry service.

## Overview

# **Properties**

*u* U parameter.*v* V parameter.

## **Special methods**

add	Add the u and v components of the other ParamUV to this ParamUV.
sub	Subtract the u and v components of a ParamUV from this ParamUV.
mu1	Multiplies the u and v components of this ParamUV by a ParamUV.
truediv	Divides the u and v components of this ParamUV by a ParamUV.
iter	Iterate a ParamUV.
repr	Represent the ParamUV as a string.

## Import detail

from ansys.geometry.core.shapes.parameterization import ParamUV

# **Property detail**

```
property ParamUV.u: ansys.geometry.core.typing.Real
     U parameter.
property ParamUV.v: ansys.geometry.core.typing.Real
     V parameter.
```

## **Method detail**

```
ParamUV.__add__(other: ParamUV) \rightarrow ParamUV
      Add the u and v components of the other ParamUV to this ParamUV.
           Parameters
               other
                   [ParamUV] The parameters to add these parameters.
           Returns
               ParamUV
                   The sum of the parameters.
ParamUV.__sub__(other: ParamUV) \rightarrow ParamUV
      Subtract the u and v components of a ParamUV from this ParamUV.
           Parameters
               other
                   [ParamUV] The parameters to subtract from these parameters.
           Returns
               ParamUV
                   The difference of the parameters.
ParamUV.__mul__(other: ParamUV) \rightarrow ParamUV
      Multiplies the u and v components of this ParamUV by a ParamUV.
           Parameters
               other
                   [ParamUV] The parameters to multiply by these parameters.
           Returns
               ParamUV
                   The product of the parameters.
ParamUV.__truediv__(other: ParamUV) \rightarrow ParamUV
      Divides the u and v components of this ParamUV by a ParamUV.
           Parameters
               other
                   [ParamUV] The parameters to divide these parameters by.
           Returns
               ParamUV
                   The quotient of the parameters.
ParamUV.__iter__()
      Iterate a ParamUV.
```

ParamUV.\_\_repr\_\_()  $\rightarrow$  str

Represent the ParamUV as a string.

## **Interval**

Interval class that defines a range of values.

## **Parameters**

start

[Real] Start value of the interval.

end

[Real] End value of the interval.

## **Overview**

## **Methods**

is_open	If the interval is open (-inf, inf).
is_closed	If the interval is closed. Neither value is inf or -inf.
is_empty	Check if the current interval is empty.
get_span	Get the quantity contained by the interval.
<pre>get_relative_val</pre>	Get an evaluation property of the interval, used in BoxUV.
is_negative	Indicate whether the current interval is negative.
self_unite	Get the union of two intervals and update the current interval.
self_intersect	Get the intersection of two intervals and update the current one.
contains_value	Check if the current interval contains the value t.
contains	Check if interval contains value t using default accuracy.
inflate	Enlarge the current interval by the given delta value.

# **Properties**

start	Start value of the interval.
end	End value of the interval.

## **Attributes**

not\_empty

# Static methods

unite	Get the union of two intervals.
intersect	Get the intersection of two intervals.

# **Special methods**

eq	Compare two intervals.
repr	Represent the interval as a string.

## Import detail

```
from ansys.geometry.core.shapes.parameterization import Interval
```

## **Property detail**

property Interval.end: ansys.geometry.core.typing.Real

End value of the interval.

## **Attribute detail**

Interval.not\_empty = True

#### Method detail

```
Interval.__eq__(other: object)
```

Compare two intervals.

Interval.is\_open()  $\rightarrow$  bool

If the interval is open (-inf, inf).

#### Returns

bool

True if both ends of the interval are negative and positive infinity respectively.

Interval.is\_closed()  $\rightarrow$  bool

If the interval is closed. Neither value is inf or -inf.

## Returns

bool

True if neither bound of the interval is infinite.

Interval.is\_empty()  $\rightarrow$  bool

Check if the current interval is empty.

## Returns

bool

True when the interval is empty, False otherwise.

 $\textbf{Interval.get\_span()} \rightarrow ansys.geometry.core.typing.Real$ 

Get the quantity contained by the interval.

The interval must be closed.

## Returns

Real

The difference between the end and start of the interval.

Interval.get\_relative\_val(t: ansys.geometry.core.typing.Real)  $\rightarrow$  ansys.geometry.core.typing.Real Get an evaluation property of the interval, used in BoxUV.

## Parameters

```
[Real] Offset to evaluate the interval at.
           Returns
                Real
                    Actual value according to the offset.
Interval.is_negative(tolerance: ansys.geometry.core.typing.Real) \rightarrow bool
      Indicate whether the current interval is negative.
           Parameters
                tolerance
                    [Real] Accepted range because the data type of the interval could be in doubles.
           Returns
                bool
                    True if the interval is negative, False otherwise.
static Interval.unite(first: Interval, second: Interval) \rightarrow Interval
      Get the union of two intervals.
           Parameters
                first
                     [Interval] First interval.
                second
                    [Interval] Second interval.
           Returns
                Interval
                    Union of the two intervals.
Interval.self_unite(other: Interval) \rightarrow None
      Get the union of two intervals and update the current interval.
           Parameters
                other
                     [Interval] Interval to unite with.
static Interval.intersect(first: Interval, second: Interval, tolerance: ansys.geometry.core.typing.Real) \rightarrow
      Get the intersection of two intervals.
           Parameters
                first
                    [Interval] First interval.
                second
                    [Interval] Second interval.
           Returns
                Interval
                    Intersection of the two intervals.
```

```
Interval.self_intersect(other: Interval, tolerance: ansys.geometry.core.typing.Real) \rightarrow None
      Get the intersection of two intervals and update the current one.
           Parameters
                other
                    [Interval] Interval to intersect with.
                tolerance
                    [Real] Accepted range of error given that the interval could be in float values.
Interval.contains_value(t: ansys.geometry.core.typing.Real, accuracy: ansys.geometry.core.typing.Real) \rightarrow
      Check if the current interval contains the value t.
           Parameters
                    [Real] Value of interest.
                accuracy
                    [Real] Accepted range of error given that the interval could be in float values.
           Returns
                bool
                    True if the interval contains the value, False otherwise.
Interval.contains(t: ansys.geometry.core.typing.Real) \rightarrow bool
      Check if interval contains value t using default accuracy.
           Parameters
                t
                    [Real] Value of interest.
           Returns
                bool
                    True if the interval contains the value, False otherwise.
Interval.inflate(delta: ansys.geometry.core.typing.Real) \rightarrow Interval
      Enlarge the current interval by the given delta value.
Interval.__repr__() \rightarrow str
      Represent the interval as a string.
Parameterization
class ansys.geometry.core.shapes.parameterization.Parameterization(form: ParamForm, type:
                                                                                       ParamType, interval:
                                                                                       Interval)
      Parameterization class describes the parameters of a specific geometry.
           Parameters
                form
                    [ParamForm] Form of the parameterization.
                type
                    [ParamType] Type of the parameterization.
```

### interval

[Interval] Interval of the parameterization.

## **Overview**

## **Properties**

form	The form of the parameterization.
type	The type of the parameterization.
interval	The interval of the parameterization.

## **Special methods**

 $\underline{\hspace{0.3cm}} \textbf{repr}\underline{\hspace{0.3cm}} \textbf{Represent the Parameterization as a string.}$ 

## Import detail

from ansys.geometry.core.shapes.parameterization import Parameterization

# **Property detail**

property Parameterization.form: ParamForm

The form of the parameterization.

property Parameterization.type: ParamType

The type of the parameterization.

property Parameterization.interval: Interval

The interval of the parameterization.

## **Method detail**

Parameterization.\_\_repr\_\_()  $\rightarrow$  str

Represent the Parameterization as a string.

### **ParamForm**

class ansys.geometry.core.shapes.parameterization.ParamForm(\*args, \*\*kwds)

Bases: enum. Enum

ParamForm enum class that defines the form of a Parameterization.

## Overview

## **Attributes**

OPEN
CLOSED
PERIODIC
OTHER

# Import detail

```
from ansys.geometry.core.shapes.parameterization import ParamForm
```

## **Attribute detail**

```
ParamForm.OPEN = 1
ParamForm.CLOSED = 2
ParamForm.PERIODIC = 3
ParamForm.OTHER = 4
```

## ParamType

```
\begin{tabular}{ll} \textbf{class} & ansys. \texttt{geometry.core.shapes.parameterization.ParamType(*} args, **kwds) \\ & Bases: & enum.Enum \\ \end{tabular}
```

ParamType enum class that defines the type of a Parameterization.

## Overview

## **Attributes**

LINEAR CIRCULAR OTHER

# Import detail

from ansys.geometry.core.shapes.parameterization import ParamType

## **Attribute detail**

```
ParamType.LINEAR = 1
ParamType.CIRCULAR = 2
ParamType.OTHER = 3
```

# **Description**

Provides the parametrization-related classes.

## **Description**

Provides the PyAnsys Geometry geometry subpackage.

## The sketch package

## **Summary**

#### **Submodules**

arc	Provides for creating and managing an arc.
box	Provides for creating and managing a box (quadrilateral).
circle	Provides for creating and managing a circle.
edge	Provides for creating and managing an edge.
ellipse	Provides for creating and managing an ellipse.
face	Provides for creating and managing a face (closed 2D sketch).
gears	Module for creating and managing gears.
polygon	Provides for creating and managing a polygon.
segment	Provides for creating and managing a segment.
sketch	Provides for creating and managing a sketch.
slot	Provides for creating and managing a slot.
trapezoid	Provides for creating and managing a trapezoid.
triangle	Provides for creating and managing a triangle.

# The arc.py module

# **Summary**

## **Classes**

Arc Provides for modeling an arc.

#### Arc

class ansys.geometry.core.sketch.arc.Arc(start: ansys.geometry.core.math.point.Point2D, end: ansys.geometry.core.math.point.Point2D, center: ansys.geometry.core.math.point.Point2D, clockwise: bool = False)

Bases: ansys.geometry.core.sketch.edge.SketchEdge

Provides for modeling an arc.

## **Parameters**

## start

[Point2D] Starting point of the arc.

#### end

[Point2D] Ending point of the arc.

## center

[Point2D] Center point of the arc.

#### clockwise

[bool, default: False] Whether the arc spans the clockwise angle between the start and end points. When False (default), the arc spans the counter-clockwise angle. When True, the arc spands the clockwise angle.

## **Overview**

## **Constructors**

from_three_points	Create an arc from three given points.
<pre>from_start_end_and_radius</pre>	Create an arc from a starting point, an ending point, and a radius.
<pre>from_start_center_and_angle</pre>	Create an arc from a starting point, a center point, and an angle.

# **Properties**

start	Starting point of the arc line.
end	Ending point of the arc line.
center	Center point of the arc.
length	Length of the arc.
radius	Radius of the arc.
angle	Angle of the arc.
is_clockwise	Flag indicating whether the rotation of the angle is clockwise.
sector_area	Area of the sector of the arc.
visualization_polydata	VTK polydata representation for PyVista visualization.

# **Special methods**

eq	Equals operator for the Arc class.
ne	Not equals operator for the Arc class.

## Import detail

```
from ansys.geometry.core.sketch.arc import Arc
```

## **Property detail**

```
property Arc.start: ansys.geometry.core.math.point.Point2D
```

Starting point of the arc line.

property Arc.end: ansys.geometry.core.math.point.Point2D

Ending point of the arc line.

property Arc.center: ansys.geometry.core.math.point.Point2D

Center point of the arc.

property Arc.length: pint.Quantity

Length of the arc.

property Arc.radius: pint.Quantity

Radius of the arc.

property Arc.angle: pint.Quantity

Angle of the arc.

property Arc.is\_clockwise: bool

Flag indicating whether the rotation of the angle is clockwise.

### **Returns**

bool

True if the sense of rotation is clockwise. False if the sense of rotation is counter-clockwise.

```
property Arc.sector_area: pint.Quantity
```

Area of the sector of the arc.

## property Arc.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

#### Returns

```
pyvista.PolyData
```

VTK pyvista. Polydata configuration.

#### **Notes**

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

## **Method detail**

```
Arc.__eq__(other: Arc) \rightarrow bool
      Equals operator for the Arc class.
Arc.__ne__(other: Arc) \rightarrow bool
```

Not equals operator for the Arc class.

**classmethod** Arc.**from\_three\_points**(start: ansys.geometry.core.math.point.Point2D, inter: ansys.geometry.core.math.point.Point2D, end:

ansys.geometry.core.math.point.Point2D)

Create an arc from three given points.

#### **Parameters**

start

[Point2D] Starting point of the arc.

inter

[Point2D] Intermediate point (location) of the arc.

end

[Point2D] Ending point of the arc.

#### Returns

Arc

Arc generated from the three points.

# **classmethod** Arc.**from\_start\_end\_and\_radius**(start: ansys.geometry.core.math.point.Point2D, end: ansys.geometry.core.math.point.Point2D, radius:

pint.Quantity |

ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, convex\_arc: bool = False,

clockwise: bool = False)

Create an arc from a starting point, an ending point, and a radius.

## **Parameters**

start

[Point2D] Starting point of the arc.

#### end

[Point2D] Ending point of the arc.

#### radius

[Quantity | Distance | Real] Radius of the arc.

# convex\_arc

[bool, default: False] Whether the arc is convex. The default is False. When False, the arc is concave. When True, the arc is convex.

#### clockwise

[bool, default: False] Whether the arc spans the clockwise angle between the start and end points. When False, the arc spans the counter-clockwise angle. When True, the arc spands the clockwise angle.

#### **Returns**

#### Arc

Arc generated from the three points.

classmethod Arc.from\_start\_center\_and\_angle(start: ansys.geometry.core.math.point.Point2D, center: ansys.geometry.core.math.point.Point2D, angle: ansys.geometry.core.misc.measurements.Angle | pint.Quantity | ansys.geometry.core.typing.Real, clockwise:

Create an arc from a starting point, a center point, and an angle.

#### **Parameters**

#### start

[Point2D] Starting point of the arc.

### center

[Point2D] Center point of the arc.

## angle

[Angle | Quantity | Real] Angle of the arc.

#### clockwise

[bool, default: False] Whether the provided angle should be considered clockwise. When False, the angle is considered counter-clockwise. When True, the angle is considered clockwise.

## Returns

### Arc

Arc generated from the three points.

## **Description**

Provides for creating and managing an arc.

## The box.py module

## **Summary**

### **Classes**

**Box** Provides for modeling a box.

#### Box

 $\textbf{class} \ \ ansys. geometry. core. sketch. box. \textbf{\textit{Box}} (\textit{center}: \ ansys. geometry. core. math. point. Point 2D, \textit{width}: \\$ 

pint.Quantity

ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0)

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides for modeling a box.

#### **Parameters**

center: Point2D

Center point of the box.

width

[Quantity | Distance | Real] Width of the box.

height

[Quantity | Distance | Real] Height of the box.

angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

## Overview

## **Properties**

center	Center point of the box.
width	Width of the box.
height	Height of the box.
perimeter	Perimeter of the box.
area	Area of the box.
visualization_polydata	VTK polydata representation for PyVista visualization.

## Import detail

```
from ansys.geometry.core.sketch.box import Box
```

## **Property detail**

```
property Box.center: ansys.geometry.core.math.point.Point2D
```

Center point of the box.

property Box.width: pint.Quantity

Width of the box.

property Box.height: pint.Quantity

Height of the box.

```
property Box.perimeter: pint.Quantity
```

Perimeter of the box.

# property Box.area: pint.Quantity

Area of the box.

### property Box.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global cartesian coordinate system.

#### **Returns**

```
pyvista.PolyData
```

VTK pyvista.Polydata configuration.

## **Description**

Provides for creating and managing a box (quadrilateral).

## The circle.py module

## **Summary**

## **Classes**

SketchCircle Provides for modeling a circle.

## SketchCircle

**class** ansys.geometry.core.sketch.circle.**SketchCircle**(*center*:

ansys.geometry.core.math.point.Point2D, radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, plane: ansys.geometry.core.math.plane.Plane = Plane())

 $Bases: \ ansys. geometry. core. sketch. face. Sketch Face, \ ansys. geometry. core. shapes. curves. \\ circle. Circle$ 

Provides for modeling a circle.

## **Parameters**

# center: Point2D

Center point of the circle.

### radius

[Quantity | Distance | Real] Radius of the circle.

### plane

[Plane, optional] Plane containing the sketched circle, which is the global XY plane by default.

## **Overview**

## **Methods**

plane_change	Redefine the plane containing the SketchCircle objects.
--------------	---------------------------------------------------------

## **Properties**

center	Center of the circle.
perimeter	Perimeter of the circle.
visualization_polydata	VTK polydata representation for PyVista visualization.

# Import detail

```
from ansys.geometry.core.sketch.circle import SketchCircle
```

## **Property detail**

```
property SketchCircle.center: ansys.geometry.core.math.point.Point2D
```

Center of the circle.

```
property SketchCircle.perimeter: pint.Quantity
```

Perimeter of the circle.

## **Notes**

This property resolves the dilemma between using the SkethFace.perimeter property and the Circle. perimeter property.

# property SketchCircle.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

#### Returns

```
pyvista.PolyData
```

VTK pyvista. Polydata configuration.

## **Method detail**

SketchCircle.plane\_change(plane: ansys.geometry.core.math.plane.Plane)  $\rightarrow$  None

Redefine the plane containing the SketchCircle objects.

## **Parameters**

## plane

[Plane] Desired new plane that is to contain the sketched circle.

## **Notes**

This implies that their 3D definition might suffer changes.

# **Description**

Provides for creating and managing a circle.

## The edge.py module

# **Summary**

#### Classes

SketchEdge Provides for modeling edges forming sketched shapes.

## SketchEdge

class ansys.geometry.core.sketch.edge.SketchEdge

Provides for modeling edges forming sketched shapes.

### Overview

## **Methods**

plane_change I	Redefine the plane containing SketchEdge objects.
----------------	---------------------------------------------------

# **Properties**

start	Starting point of the edge.
end	Ending point of the edge.
length	Length of the edge.
visualization_polydata	VTK polydata representation for PyVista visualization.

## Import detail

from ansys.geometry.core.sketch.edge import SketchEdge

# **Property detail**

property SketchEdge.start: ansys.geometry.core.math.point.Point2D

## Abstractmethod

Starting point of the edge.

property SketchEdge.end: ansys.geometry.core.math.point.Point2D

## Abstractmethod

Ending point of the edge.

property SketchEdge.length: pint.Quantity

## Abstractmethod

Length of the edge.

## property SketchEdge.visualization\_polydata: pyvista.PolyData

#### Abstractmethod

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

#### Returns

```
pyvista.PolyData
```

VTK pyvista. Polydata configuration.

### Method detail

 $\label{eq:sketchedge.plane} SketchEdge.plane\_change(plane: ansys.geometry.core.math.plane.Plane) \rightarrow None$ 

Redefine the plane containing SketchEdge objects.

#### **Parameters**

## plane

[Plane] Desired new plane that is to contain the sketched edge.

#### **Notes**

This implies that their 3D definition might suffer changes. By default, this metho does nothing. It is required to be implemented in child SketchEdge classes.

# **Description**

Provides for creating and managing an edge.

## The ellipse.py module

## **Summary**

### **Classes**

SketchEllipse Provides for modeling an ellipse.

### SketchEllipse

**class** ansys.geometry.core.sketch.ellipse.**SketchEllipse**(center:

ansys.geometry.core.math.point.Point2D,
major\_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance
| ansys.geometry.core.typing.Real,
minor\_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance
| ansys.geometry.core.typing.Real, angle:
pint.Quantity | ansys.geometry.core.misc.measurements.Angle
| ansys.geometry.core.misc.measurements.Angle
| ansys.geometry.core.typing.Real = 0,
plane:
ansys.geometry.core.math.plane.Plane =
Plane())

Bases: ansys.geometry.core.sketch.face.SketchFace, ansys.geometry.core.shapes.curves.ellipse.Ellipse

Provides for modeling an ellipse.

## **Parameters**

#### center: Point2D

Center point of the ellipse.

## major\_radius

[Quantity | Distance | Real] Major radius of the ellipse.

#### minor radius

[Quantity | Distance | Real] Minor radius of the ellipse.

### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

## plane

[Plane, optional] Plane containing the sketched ellipse, which is the global XY plane by default.

#### Overview

## Methods

plane_change	Redefine the plane containing SketchEllipse objects.

## **Properties**

center	Center point of the ellipse.
angle	Orientation angle of the ellipse.
perimeter	Perimeter of the circle.
visualization_polydata	VTK polydata representation for PyVista visualization.

# Import detail

```
from ansys.geometry.core.sketch.ellipse import SketchEllipse
```

## **Property detail**

```
property SketchEllipse.center: ansys.geometry.core.math.point.Point2D
```

Center point of the ellipse.

```
property SketchEllipse.angle: pint.Quantity
```

Orientation angle of the ellipse.

```
property SketchEllipse.perimeter: pint.Quantity
```

Perimeter of the circle.

## **Notes**

This property resolves the dilemma between using the SkethFace.perimeter property and the Ellipse. perimeter property.

## property SketchEllipse.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

#### Returns

```
pyvista.PolyData
```

VTK pyvista.Polydata configuration.

#### Method detail

 $SketchEllipse.\textbf{plane\_change}(plane: ansys.geometry.core.math.plane.Plane) \rightarrow None$ 

Redefine the plane containing SketchEllipse objects.

#### **Parameters**

#### plane

[Plane] Desired new plane that is to contain the sketched ellipse.

#### **Notes**

This implies that their 3D definition might suffer changes.

## **Description**

Provides for creating and managing an ellipse.

## The face.py module

## **Summary**

#### **Classes**

SketchFace Provides for modeling a face.

#### SketchFace

class ansys.geometry.core.sketch.face.SketchFace

Provides for modeling a face.

### Overview

## **Methods**

plane\_change Redefine the plane containing SketchFace objects.

## **Properties**

edges	List of all component edges forming the face.
perimeter	Perimeter of the face.
visualization_polydata	VTK polydata representation for PyVista visualization.

## Import detail

```
from ansys.geometry.core.sketch.face import SketchFace
```

## **Property detail**

```
property SketchFace.edges: list[ansys.geometry.core.sketch.edge.SketchEdge]
```

List of all component edges forming the face.

```
property SketchFace.perimeter: pint.Quantity
```

Perimeter of the face.

# property SketchFace.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

#### Returns

```
pyvista.PolyData
```

VTK pyvista. Polydata configuration.

## **Method detail**

SketchFace.plane\_change(plane: ansys.geometry.core.math.plane.Plane)  $\rightarrow$  None

Redefine the plane containing SketchFace objects.

### **Parameters**

#### plane

[Plane] Desired new plane that is to contain the sketched face.

#### **Notes**

This implies that their 3D definition might suffer changes. This method does nothing by default. It is required to be implemented in child SketchFace classes.

## **Description**

Provides for creating and managing a face (closed 2D sketch).

## The gears.py module

## **Summary**

## **Classes**

Gear	Provides the base class for sketching gears.
DummyGear	Provides the dummy class for sketching gears.
SpurGear	Provides the class for sketching spur gears.

## Gear

class ansys.geometry.core.sketch.gears.Gear

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides the base class for sketching gears.

#### Overview

## **Properties**

visualization\_polydata VTK polydata representation for PyVista visualization.

# Import detail

from ansys.geometry.core.sketch.gears import Gear

## **Property detail**

## property Gear.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

## Returns

```
pyvista.PolyData
```

VTK pyvista.Polydata configuration.

## DummyGear

class ansys.geometry.core.sketch.gears.DummyGear(origin: ansys.geometry.core.math.point.Point2D,

outer radius: pint.Quantity |

 $ansys.geometry.core.misc.measurements.Distance \mid$ 

ansys.geometry.core.typing.Real, inner\_radius:

pint.Quantity |

ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, n\_teeth: int)

Bases: Gear

Provides the dummy class for sketching gears.

#### **Parameters**

#### origin

[Point2D] Origin of the gear.

## outer\_radius

[Quantity | Distance | Real] Outer radius of the gear.

### inner radius

[Quantity | Distance | Real] Inner radius of the gear.

## n\_teeth

[int] Number of teeth of the gear.

## Import detail

```
from ansys.geometry.core.sketch.gears import DummyGear
```

## **SpurGear**

 $\textbf{class} \ \ ansys. geometry. core. sketch. gears. \textbf{SpurGear} (\textit{origin:} \ ansys. geometry. core. math. point. Point 2D, ansys. geometry. core. math. point 2D, ansys. geometry. geometry. The ansys. ge$ 

module: ansys.geometry.core.typing.Real, pressure\_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, n\_teeth: int)

Bases: Gear

Provides the class for sketching spur gears.

## **Parameters**

### origin

[Point2D] Origin of the spur gear.

#### module

[Real] Module of the spur gear. This is also the ratio between the pitch circle diameter in millimeters and the number of teeth.

#### pressure angle

[Quantity | Angle | Real] Pressure angle of the spur gear.

## n\_teeth

[int] Number of teeth of the spur gear.

#### **Overview**

## **Properties**

origin	Origin of the spur gear.
module	Module of the spur gear.
<pre>pressure_angle</pre>	Pressure angle of the spur gear.
n_teeth	Number of teeth of the spur gear.
ref_diameter	Reference diameter of the spur gear.
base_diameter	Base diameter of the spur gear.
addendum	Addendum of the spur gear.
dedendum	Dedendum of the spur gear.
tip_diameter	Tip diameter of the spur gear.
root_diameter	Root diameter of the spur gear.

# Import detail

```
from ansys.geometry.core.sketch.gears import SpurGear
```

## **Property detail**

```
property SpurGear.origin: ansys.geometry.core.math.point.Point2D
    Origin of the spur gear.
```

property SpurGear.pressure\_angle: pint.Quantity
 Pressure angle of the spur gear.

property SpurGear.n\_teeth: int
 Number of teeth of the spur gear.

property SpurGear.ref\_diameter: ansys.geometry.core.typing.Real
 Reference diameter of the spur gear.

property SpurGear.addendum: ansys.geometry.core.typing.Real
 Addendum of the spur gear.

property SpurGear.dedendum: ansys.geometry.core.typing.Real
 Dedendum of the spur gear.

property SpurGear.root\_diameter: ansys.geometry.core.typing.Real
 Root diameter of the spur gear.

# **Description**

Module for creating and managing gears.

The polygon.py module

Summary

**Classes** 

Polygon Provides for modeling regular polygons.

## Polygon

class ansys.geometry.core.sketch.polygon.Polygon(center: ansys.geometry.core.math.point.Point2D,

inner\_radius: pint.Quantity |

ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, sides: int, angle:

pint.Quantity |

ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0)

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides for modeling regular polygons.

#### **Parameters**

#### center: Point2D

Center point of the circle.

#### inner radius

[Quantity | Distance | Real] Inner radius (apothem) of the polygon.

#### sides

[int] Number of sides of the polygon.

#### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

## Overview

## **Properties**

center	Center point of the polygon.
inner_radius	Inner radius (apothem) of the polygon.
n_sides	Number of sides of the polygon.
angle	Orientation angle of the polygon.
length	Side length of the polygon.
outer_radius	Outer radius of the polygon.
perimeter	Perimeter of the polygon.
area	Area of the polygon.
visualization_polydata	VTK polydata representation for PyVista visualization.

# Import detail

```
from ansys.geometry.core.sketch.polygon import Polygon
```

## **Property detail**

```
property Polygon.center: ansys.geometry.core.math.point.Point2D
```

Center point of the polygon.

```
property Polygon.inner_radius: pint.Quantity
```

Inner radius (apothem) of the polygon.

# property Polygon.n\_sides: int

Number of sides of the polygon.

```
property Polygon.angle: pint.Quantity
     Orientation angle of the polygon.
property Polygon.length: pint.Quantity
     Side length of the polygon.
property Polygon.outer_radius: pint.Quantity
     Outer radius of the polygon.
property Polygon.perimeter: pint.Quantity
     Perimeter of the polygon.
property Polygon.area: pint.Quantity
     Area of the polygon.
property Polygon.visualization_polydata: pyvista.PolyData
     VTK polydata representation for PyVista visualization.
     The representation lies in the X/Y plane within the standard global Cartesian coordinate system.
          Returns
              pyvista.PolyData
                  VTK pyvista. Polydata configuration.
Description
Provides for creating and managing a polygon.
The segment.py module
Summary
Classes
                        SketchSegment
                                          Provides segment representation of a line.
SketchSegment
class ansys.geometry.core.sketch.segment.SketchSegment(start:
                                                               ansys.geometry.core.math.point.Point2D,
                                                               ansys.geometry.core.math.point.Point2D,
                                                               ansys.geometry.core.math.plane.Plane =
                                                               Plane())
     Bases: ansys.geometry.core.sketch.edge.SketchEdge, ansys.geometry.core.shapes.curves.
     line.Line
     Provides segment representation of a line.
          Parameters
              start
                  [Point2D] Starting point of the line segment.
              end
```

[Point2D] Ending point of the line segment.

### plane

[Plane, optional] Plane containing the sketched circle, which is the global XY plane by default.

## **Overview**

## Methods

plane_change	Redefine the plane containing SketchSegment objects.
--------------	------------------------------------------------------

## **Properties**

start	Starting point of the segment.
end	Ending point of the segment.
length	Length of the segment.
visualization_polydata	VTK polydata representation for PyVista visualization.

## **Special methods**

eq	Equals operator for the SketchSegment class.
ne	Not equals operator for the SketchSegment class.

## Import detail

```
from ansys.geometry.core.sketch.segment import SketchSegment
```

## **Property detail**

```
property SketchSegment.start: ansys.geometry.core.math.point.Point2D
    Starting point of the segment.
```

```
property SketchSegment.length: pint.Quantity
```

Length of the segment.

## property SketchSegment.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

## **Returns**

```
pyvista.PolyData
```

VTK pyvista.Polydata configuration.

## **Method detail**

```
{\tt SketchSegment.\_\_eq\_\_(\it other: SketchSegment) \rightarrow bool}
```

Equals operator for the SketchSegment class.

 ${\tt SketchSegment.\_\_ne\_\_}(\textit{other}: SketchSegment) \rightarrow bool$ 

Not equals operator for the SketchSegment class.

SketchSegment.plane\_change(plane: ansys.geometry.core.math.plane.Plane)  $\rightarrow$  None

Redefine the plane containing SketchSegment objects.

#### **Parameters**

### plane

[Plane] Desired new plane that is to contain the sketched segment.

#### **Notes**

This implies that their 3D definition might suffer changes.

## **Description**

Provides for creating and managing a segment.

# The sketch.py module

## **Summary**

### **Classes**

Sketch Provides for building 2D sketch elements.

## **Attributes**

SketchObject Type to refer to both SketchEdge and SketchFace.

## Sketch

Provides for building 2D sketch elements.

### Overview

#### **Methods**

translate_sketch_plane	Translate the origin location of the active sketch plane.
translate_sketch_plane_by_offset	Translate the origin location of the active sketch plane by offsets.
<pre>translate_sketch_plane_by_distance</pre>	Translate the origin location active sketch plane by distance.
get	Get a list of shapes with a given tag.
face	Add a sketch face to the sketch.
edge	Add a sketch edge to the sketch.
select	Add all objects that match provided tags to the current context.
segment	Add a segment sketch object to the sketch plane.
segment_to_point	Add a segment to the sketch plane starting from the previous end point.
<pre>segment_from_point_and_vector</pre>	Add a segment to the sketch starting from a given starting point.

continues on next page

Table 3 – continued from previous page

segment_from_vector	Add a segment to the sketch starting from the previous end point.
arc	Add an arc to the sketch plane.
arc_to_point	Add an arc to the sketch starting from the previous end point.
<pre>arc_from_three_points</pre>	Add an arc to the sketch plane from three given points.
<pre>arc_from_start_end_and_radius</pre>	Add an arc from the start, end points and a radius.
<pre>arc_from_start_center_and_angle</pre>	Add an arc from the start, center point, and angle.
triangle	Add a triangle to the sketch using given vertex points.
trapezoid	Add a trapezoid to the sketch using given vertex points.
circle	Add a circle to the plane at a given center.
box	Create a box on the sketch.
slot	Create a slot on the sketch.
ellipse	Create an ellipse on the sketch.
polygon	Create a polygon on the sketch.
dummy_gear	Create a dummy gear on the sketch.
spur_gear	Create a spur gear on the sketch.
tag	Add a tag to the active selection of sketch objects.
plot	Plot all objects of the sketch to the scene.
plot_selection	Plot the current selection to the scene.
sketch_polydata	Get polydata configuration for all objects of the sketch.
sketch_polydata_faces	Get polydata configuration for all faces of the sketch to the scene.
sketch_polydata_edges	Get polydata configuration for all edges of the sketch to the scene.

## **Properties**

plane	Sketch plane configuration.
edges	List of all independently sketched edges.
faces	List of all independently sketched faces.

# Import detail

from ansys.geometry.core.sketch.sketch import Sketch

# **Property detail**

property Sketch.plane: ansys.geometry.core.math.plane.Plane

Sketch plane configuration.

property Sketch.edges: list[ansys.geometry.core.sketch.edge.SketchEdge]

List of all independently sketched edges.

# **Notes**

Independently sketched edges are not assigned to a face. Face edges are not included in this list.

property Sketch.faces: list[ansys.geometry.core.sketch.face.SketchFace]

List of all independently sketched faces.

## **Method detail**

Sketch.translate\_sketch\_plane(translation: ansys.geometry.core.math.vector.Vector3D)  $\rightarrow$  Sketch Translate the origin location of the active sketch plane.

#### **Parameters**

#### translation

[Vector3D] Vector defining the translation. Meters is the expected unit.

#### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

### Sketch.translate\_sketch\_plane\_by\_offset(x: pint.Quantity |

ansys.geometry.core.misc.measurements.Distance =  $Quantity(0, DEFAULT\_UNITS.LENGTH)$ ,  $y: pint.Quantity \mid$  ansys.geometry.core.misc.measurements.Distance =  $Quantity(0, DEFAULT\_UNITS.LENGTH)$ ,  $z: pint.Quantity \mid$  ansys.geometry.core.misc.measurements.Distance =  $Quantity(0, DEFAULT\_UNITS.LENGTH)$ )  $\rightarrow Sketch$ 

Translate the origin location of the active sketch plane by offsets.

#### **Parameters**

x

[Quantity|Distance, default: Quantity(0, DEFAULT\_UNITS.LENGTH)] Amount to translate the origin of the x-direction.

y

[Quantity|Distance, default: Quantity(0, DEFAULT\_UNITS.LENGTH)] Amount to translate the origin of the y-direction.

Z

[Quantity|Distance, default: Quantity(0, DEFAULT\_UNITS.LENGTH)] Amount to translate the origin of the z-direction.

### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

Sketch.translate\_sketch\_plane\_by\_distance(direction: ansys.geometry.core.math.vector.UnitVector3D, distance: pint.Quantity |

ansys.geometry.core.misc.measurements.Distance)  $\rightarrow$  *Sketch* 

Translate the origin location active sketch plane by distance.

## **Parameters**

#### direction

[UnitVector3D] Direction to translate the origin.

#### distance

[Quantity | Distance] Distance to translate the origin.

## Returns

### Sketch

```
Sketch.get(tag: str) \rightarrow list[SketchObject]
      Get a list of shapes with a given tag.
            Parameters
                tag
                    [str] Tag to query against.
Sketch. face (face: ansys.geometry.core.sketch.face.SketchFace, tag: str \mid None = None) \rightarrow Sketch
      Add a sketch face to the sketch.
            Parameters
                face
                     [SketchFace] Face to add.
                tag
                     [str, default: None] User-defined label for identifying the face.
            Returns
                Sketch
                     Revised sketch state ready for further sketch actions.
Sketch.edge (edge: ansys.geometry.core.sketch.edge.SketchEdge, tag: str \mid None = None) \rightarrow Sketch
      Add a sketch edge to the sketch.
            Parameters
                edge
                     [SketchEdge] Edge to add.
                     [str, default: None] User-defined label for identifying the edge.
            Returns
                Sketch
                     Revised sketch state ready for further sketch actions.
Sketch.select(*tags: str) \rightarrow Sketch
      Add all objects that match provided tags to the current context.
Sketch.segment(start: ansys.geometry.core.math.point.Point2D, end: ansys.geometry.core.math.point2D,
                    tag: str \mid None = None) \rightarrow Sketch
      Add a segment sketch object to the sketch plane.
            Parameters
                start
                     [Point2D] Starting point of the line segment.
                end
                     [Point2D] Ending point of the line segment.
                tag
                    [str, default: None] User-defined label for identifying the edge.
            Returns
                Sketch
                     Revised sketch state ready for further sketch actions.
```

Sketch.segment\_to\_point(end: ansys.geometry.core.math.point.Point2D,  $tag: str \mid None = None$ )  $\rightarrow Sketch$ Add a segment to the sketch plane starting from the previous end point.

### **Parameters**

#### end

[Point2D] Ending point of the line segment.

tag

[str, default: None] User-defined label for identifying the edge.

#### **Returns**

#### Sketch

Revised sketch state ready for further sketch actions.

#### **Notes**

The starting point of the created edge is based upon the current context of the sketch, such as the end point of a previously added edge.

 $\label{eq:segment_from_point_and_vector} Sketch. \textbf{segment_from_point_and_vector} (start: ansys. geometry. core. math. point. Point 2D, vector: ansys. geometry. core. math. vector. Vector 2D, tag: str | None = None)$ 

Add a segment to the sketch starting from a given starting point.

#### **Parameters**

#### start

[Point2D] Starting point of the line segment.

#### vector

[Vector2D] Vector defining the line segment. Vector magnitude determines the segment endpoint. Vector magnitude is assumed to be in the same unit as the starting point.

tag

[str, default: None] User-defined label for identifying the edge.

#### **Returns**

#### Sketch

Revised sketch state ready for further sketch actions.

#### **Notes**

Vector magnitude determines the segment endpoint. Vector magnitude is assumed to use the same unit as the starting point.

Sketch.segment\_from\_vector(vector: ansys.geometry.core.math.vector.Vector2D, tag: str | None = None)
Add a segment to the sketch starting from the previous end point.

### **Parameters**

#### vector

[Vector2D] Vector defining the line segment.

tag

[str, default: None] User-defined label for identifying the edge.

## Returns

#### Sketch

#### **Notes**

The starting point of the created edge is based upon the current context of the sketch, such as the end point of a previously added edge.

Vector magnitude determines the segment endpoint. Vector magnitude is assumed to use the same unit as the starting point in the previous context.

Sketch.arc(start: ansys.geometry.core.math.point.Point2D, end: ansys.geometry.core.math.point.Point2D, clockwise: bool = False, tag:  $str \mid None = None$ )  $\rightarrow Sketch$  Add an arc to the sketch plane.

### **Parameters**

#### start

[Point2D] Starting point of the arc.

#### end

[Point2D] Ending point of the arc.

#### center

[Point2D] Center point of the arc.

#### clockwise

[bool, default: False] Whether the arc spans the angle clockwise between the start and end points. When False `` (default), the arc spans the angle counter-clockwise. When ``True, the arc spans the angle clockwise.

#### tag

[str, default: None] User-defined label for identifying the edge.

#### Returns

### Sketch

Revised sketch state ready for further sketch actions.

Sketch.arc\_to\_point(end: ansys.geometry.core.math.point.Point2D, center:

```
ansys.geometry.core.math.point.Point2D, clockwise: bool = False, tag: str \mid None = None)

\rightarrow Sketch
```

Add an arc to the sketch starting from the previous end point.

#### **Parameters**

#### end

[Point2D] Ending point of the arc.

### center

[Point2D] Center point of the arc.

### clockwise

[bool, default: False] Whether the arc spans the angle clockwise between the start and end points. When False (default), the arc spans the angle counter-clockwise. When True, the arc spans the angle clockwise.

#### tag

[str, default: None] User-defined label for identifying the edge.

## Returns

#### Sketch

## **Notes**

The starting point of the created edge is based upon the current context of the sketch, such as the end point of a previously added edge.

```
\label{eq:core.math.point.Point2D}  Sketch. \textbf{arc\_from\_three\_points} (\textit{start:} \ ansys. geometry. core. math.point. Point2D, \textit{inter:} \\ ansys. geometry. core. math.point. Point2D, \textit{end:} \\ ansys. geometry. core. math.point. Point2D, \textit{tag:} \textit{str} \mid \textit{None} = \textit{None}) \rightarrow \textit{Sketch}
```

Add an arc to the sketch plane from three given points.

```
Parameters
```

#### start

[Point2D] Starting point of the arc.

#### inter

[Point2D] Intermediate point (location) of the arc.

#### end

[Point2D] End point of the arc.

tag

[str, default: None] User-defined label for identifying the edge.

#### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

```
Sketch.arc_from_start_end_and_radius(start: ansys.geometry.core.math.point.Point2D, end:
```

ansys.geometry.core.math.point.Point2D, radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, convex\_arc: bool = False, clockwise: bool = False, tag:  $str \mid None = None$ )  $\rightarrow Sketch$ 

Add an arc from the start, end points and a radius.

# Parameters

### start

[Point2D] Starting point of the arc.

# end

[Point2D] Ending point of the arc.

#### radius

[Quantity | Distance | Real] Radius of the arc.

## convex\_arc

[bool, default: False] Whether the arc is convex. The default is False. When False, the arc spans the concave version of the arc. When True, the arc spans the convex version of the arc.

#### clockwise

[bool, default: False] Whether the arc spans the angle clockwise between the start and end points. When False, the arc spans the angle counter-clockwise. When True, the arc spans the angle clockwise.

### tag

[str, default: None] User-defined label for identifying the edge.

#### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

```
Sketch.arc_from_start_center_and_angle(start: ansys.geometry.core.math.point.Point2D, center: ansys.geometry.core.math.point.Point2D, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, clockwise: bool = False, tag: str \mid None = None) \rightarrow Sketch
```

Add an arc from the start, center point, and angle.

#### **Parameters**

#### start

[Point2D] Starting point of the arc.

#### center

[Point2D] Center point of the arc.

#### angle

[Quantity | Angle | Real] Angle of the arc.

#### clockwise

[bool, default: False] Whether the arc spans the angle clockwise. The default is False. When False, the arc spans the angle counter-clockwise. When True, the arc spans the angle clockwise.

## Returns

#### Sketch

Revised sketch state ready for further sketch actions.

Sketch.triangle(point1: ansys.geometry.core.math.point.Point2D, point2:

ansys.geometry.core.math.point.Point2D, point3: ansys.geometry.core.math.point.Point2D, tag:  $str \mid None = None) \rightarrow Sketch$ 

Add a triangle to the sketch using given vertex points.

## Parameters

### point1

[Point2D] Point that represents a vertex of the triangle.

### point2

[Point2D] Point that represents a vertex of the triangle.

## point3

[Point2D] Point that represents a vertex of the triangle.

## tag

[str, default: None] User-defined label for identifying the face.

#### Returns

### Sketch

```
Sketch.trapezoid(base_width: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, base_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, base_asymmetric_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real | None = None, center: ansys.geometry.core.math.point.Point2D = ZERO_POINT2D, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0, tag: str | None = None) → Sketch
```

Add a trapezoid to the sketch using given vertex points.

### **Parameters**

#### base width

[Quantity | Distance | Real] Width of the lower base of the trapezoid.

### height

[Quantity | Distance | Real] Height of the slot.

## base\_angle

[Quantity | Distance | Real] Angle for trapezoid generation. Represents the angle on the base of the trapezoid.

### base\_asymmetric\_angle

[Quantity | Angle | Real | None, default: None] Asymmetrical angles on each side of the trapezoid. The default is None, in which case the trapezoid is symmetrical. If provided, the trapezoid is asymmetrical and the right corner angle at the base of the trapezoid is set to the provided value.

#### center: Point2D, default: ZERO POINT2D

Center point of the trapezoid.

### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

### tag

[str, default: None] User-defined label for identifying the face.

#### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

### **Notes**

If an asymmetric base angle is defined, the base angle is applied to the left-most angle, and the asymmetric base angle is applied to the right-most angle.

```
Sketch.circle(center: ansys.geometry.core.math.point.Point2D, radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, tag: str \mid None = None) \rightarrow Sketch
```

Add a circle to the plane at a given center.

#### **Parameters**

#### center: Point2D

Center point of the circle.

### radius

[Quantity | Distance | Real] Radius of the circle.

tag

```
[str, default: None] User-defined label for identifying the face.
           Returns
                Sketch
                    Revised sketch state ready for further sketch actions.
Sketch.box(center: ansys.geometry.core.math.point.Point2D, width: pint.Quantity |
              ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height:
              pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real,
              angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle |
              ansys.geometry.core.typing.Real = 0, tag: str \mid None = None \rightarrow Sketch
      Create a box on the sketch.
           Parameters
                center: Point2D
                    Center point of the box.
                width
                    [Quantity | Distance | Real] Width of the box.
                height
                    [Quantity | Distance | Real] Height of the box.
                    [Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.
                tag
                    [str, default: None] User-defined label for identifying the face.
           Returns
                Sketch
                    Revised sketch state ready for further sketch actions.
Sketch.slot(center: ansys.geometry.core.math.point.Point2D, width: pint.Quantity |
               ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height:
               pint. Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real,
               angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle |
               ansys.geometry.core.typing.Real = 0, tag: str \mid None = None) \rightarrow Sketch
      Create a slot on the sketch.
           Parameters
                center: Point2D
                    Center point of the slot.
                width
                    [Quantity | Distance | Real] Width of the slot.
                height
                    [Quantity | Distance | Real] Height of the slot.
                angle
                    [Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.
                tag
                    [str, default: None] User-defined label for identifying the face.
           Returns
```

#### Sketch

Revised sketch state ready for further sketch actions.

```
Sketch.ellipse(center: ansys.geometry.core.math.point.Point2D, major_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, minor_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0, tag: str | None = None) \rightarrow Sketch
```

Create an ellipse on the sketch.

### **Parameters**

#### center: Point2D

Center point of the ellipse.

#### major\_radius

[Quantity | Distance | Real] Semi-major axis of the ellipse.

### minor radius

[Quantity | Distance | Real] Semi-minor axis of the ellipse.

#### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

### tag

[str, default: None] User-defined label for identifying the face.

#### Returns

#### Sketch

Revised sketch state ready for further sketch actions.

```
Sketch.polygon(center: ansys.geometry.core.math.point.Point2D, inner_radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, sides: int, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0, tag: str \mid None = None) \rightarrow Sketch
```

Create a polygon on the sketch.

#### **Parameters**

#### center: Point2D

Center point of the polygon.

### inner\_radius

[Quantity | Distance | Real] Inner radius (apothem) of the polygon.

#### sides

[int] Number of sides of the polygon.

### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

#### tag

[str, default: None] User-defined label for identifying the face.

### Returns

#### Sketch

```
Sketch.dummy_gear(origin: ansys.geometry.core.math.point.Point2D, outer_radius: pint.Quantity |
                       ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real,
                       inner radius: pint.Quantity | ansys.geometry.core.misc.measurements.Distance |
                       ansys.geometry.core.typing.Real, n_teeth: int, tag: str \mid None = None) \rightarrow Sketch
      Create a dummy gear on the sketch.
           Parameters
                origin
                    [Point2D] Origin of the gear.
                outer radius
                    [Quantity | Distance | Real] Outer radius of the gear.
                inner radius
                    [Quantity | Distance | Real] Inner radius of the gear.
                n teeth
                    [int] Number of teeth of the gear.
                tag
                    [str, default: None] User-defined label for identifying the face.
           Returns
                Sketch
                    Revised sketch state ready for further sketch actions.
Sketch.spur_gear(origin: ansys.geometry.core.math.point2D, module: ansys.geometry.core.typing.Real,
                      pressure_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle |
                      ansys.geometry.core.typing.Real, n_teeth: int, tag: str \mid None = None \rightarrow Sketch
      Create a spur gear on the sketch.
           Parameters
                origin
                    [Point2D] Origin of the spur gear.
                module
                    [Real] Module of the spur gear. This is also the ratio between the pitch circle diameter in
                    millimeters and the number of teeth.
                pressure_angle
                    [Quantity | Angle | Real] Pressure angle of the spur gear.
                    [int] Number of teeth of the spur gear.
                tag
                    [str, default: None] User-defined label for identifying the face.
           Returns
                Sketch
                    Revised sketch state ready for further sketch actions.
Sketch.tag(tag: str) \rightarrow None
      Add a tag to the active selection of sketch objects.
           Parameters
                tag
```

[str] Tag to assign to the sketch objects.

Sketch.plot(view\_2d: bool = False, screenshot: str | None = None, use\_trame: bool | None = None, selected\_pd\_objects: list[pyvista.PolyData] = None, \*\*plotting\_options: dict | None)

Plot all objects of the sketch to the scene.

#### **Parameters**

#### view 2d

[bool, default: False] Whether to represent the plot in a 2D format.

#### screenshot

[str, optional] Path for saving a screenshot of the image that is being represented.

#### use trame

[bool, default: None] Whether to enables the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

### \*\*plotting options

[dict, optional] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

Sketch.plot\_selection(view\_2d: bool = False, screenshot: str | None = None, use\_trame: bool | None = None, \*\*plotting\_options: dict | None)

Plot the current selection to the scene.

#### **Parameters**

#### view 2d

[bool, default: False] Whether to represent the plot in a 2D format.

#### screenshot

[str, optional] Path for saving a screenshot of the image that is being represented.

#### use trame

[bool, default: None] Whether to enables the use of trame. The default is None, in which case the ansys.tools.visualization\_interface.USE\_TRAME global setting is used.

## \*\*plotting\_options

[dict, optional] Keyword arguments for plotting. For allowable keyword arguments, see the Plotter.add\_mesh method.

## Sketch.sketch\_polydata() $\rightarrow$ list[pyvista.PolyData]

Get polydata configuration for all objects of the sketch.

### Returns

### list[PolyData]

List of the polydata configuration for all edges and faces in the sketch.

## Sketch.sketch\_polydata\_faces() → list[pyvista.PolyData]

Get polydata configuration for all faces of the sketch to the scene.

### Returns

## list[PolyData]

List of the polydata configuration for faces in the sketch.

## Sketch.sketch\_polydata\_edges() $\rightarrow$ list[pyvista.PolyData]

Get polydata configuration for all edges of the sketch to the scene.

#### Returns

#### list[PolyData]

List of the polydata configuration for edges in the sketch.

# **Description**

Provides for creating and managing a sketch.

#### Module detail

## sketch.SketchObject

Type to refer to both SketchEdge and SketchFace.

### The slot.py module

### **Summary**

#### **Classes**

*Slot* Provides for modeling a 2D slot.

#### Slot

**class** ansys.geometry.core.sketch.slot.**Slot**(*center*: ansys.geometry.core.math.point.Point2D, *width*:

pint.Quantity |

ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0)

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides for modeling a 2D slot.

#### **Parameters**

center: :class:`Point2D <ansys.geometry.core.math.point.Point2D>`

Center point of the slot.

## width

[Quantity | Distance | Real] Width of the slot main body.

#### height

[Quantity | Distance | Real] Height of the slot.

#### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

#### Overview

# **Properties**

center	Center of the slot.
width	Width of the slot.
height	Height of the slot.
perimeter	Perimeter of the slot.
area	Area of the slot.
visualization_polydata	VTK polydata representation for PyVista visualization.

# Import detail

```
from ansys.geometry.core.sketch.slot import Slot
```

## **Property detail**

Height of the slot.

property Slot.perimeter: pint.Quantity

Perimeter of the slot.

property Slot.area: pint.Quantity

Area of the slot.

property Slot.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

### Returns

```
pyvista.PolyData
VTK pyvista.Polydata configuration.
```

# **Description**

Provides for creating and managing a slot.

The trapezoid.py module

**Summary** 

**Classes** 

*Trapezoid* Provides for modeling a 2D trapezoid.

Trapezoid

**class** ansys.geometry.core.sketch.trapezoid.**Trapezoid**(base\_width: pint.Quantity | an-

sys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, height: pint.Quantity | ansys.geometry.core.misc.measurements.Distance | ansys.geometry.core.typing.Real, base angle: pint.Quantity ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real, base\_asymmetric\_angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real | None = None, center: ansys.geometry.core.math.point.Point2D = ZERO\_POINT2D, angle: pint.Quantity | ansys.geometry.core.misc.measurements.Angle | ansys.geometry.core.typing.Real = 0)

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides for modeling a 2D trapezoid.

#### **Parameters**

#### base width

[Quantity | Distance | Real] Width of the lower base of the trapezoid.

#### height

[Quantity | Distance | Real] Height of the slot.

#### base angle

[Quantity | Distance | Real] Angle for trapezoid generation. Represents the angle on the base of the trapezoid.

### base\_asymmetric\_angle

[Quantity | Angle | Real | None, default: None] Asymmetrical angles on each side of the trapezoid. The default is None, in which case the trapezoid is symmetrical. If provided, the trapezoid is asymmetrical and the right corner angle at the base of the trapezoid is set to the provided value.

### center: Point2D, default: ZERO\_POINT2D

Center point of the trapezoid.

### angle

[Quantity | Angle | Real, default: 0] Placement angle for orientation alignment.

## **Notes**

If an asymmetric base angle is defined, the base angle is applied to the left-most angle, and the asymmetric base angle is applied to the right-most angle.

### **Overview**

## **Properties**

center	Center of the trapezoid.
base_width	Width of the trapezoid.
height	Height of the trapezoid.
visualization_polydata	VTK polydata representation for PyVista visualization.

## Import detail

```
from ansys.geometry.core.sketch.trapezoid import Trapezoid
```

## **Property detail**

property Trapezoid.center: ansys.geometry.core.math.point.Point2D

Center of the trapezoid.

property Trapezoid.base\_width: pint.Quantity

Width of the trapezoid.

property Trapezoid.height: pint.Quantity

Height of the trapezoid.

property Trapezoid.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

#### **Returns**

pyvista.PolyData

VTK pyvista. Polydata configuration.

## **Description**

Provides for creating and managing a trapezoid.

## The triangle.py module

### **Summary**

#### Classes

Triangle Provides for modeling 2D triangles.
----------------------------------------------

## **Triangle**

class ansys.geometry.core.sketch.triangle.Triangle(point1: ansys.geometry.core.math.point.Point2D,

point2: ansys.geometry.core.math.point.Point2D,

*point3:* ansys.geometry.core.math.point.Point2D)

Bases: ansys.geometry.core.sketch.face.SketchFace

Provides for modeling 2D triangles.

#### **Parameters**

### point1: Point2D

Point that represents a triangle vertex.

#### point2: Point2D

Point that represents a triangle vertex.

### point3: Point2D

Point that represents a triangle vertex.

## Overview

## **Properties**

point1	Triangle vertex 1.
point2	Triangle vertex 2.
point3	Triangle vertex 3.
visualization_polydata	VTK polydata representation for PyVista visualization.

# Import detail

```
from ansys.geometry.core.sketch.triangle import Triangle
```

### **Property detail**

```
property Triangle.point1: ansys.geometry.core.math.point.Point2D
    Triangle vertex 1.
```

```
property Triangle.point3: ansys.geometry.core.math.point.Point2D
    Triangle vertex 3.
```

### property Triangle.visualization\_polydata: pyvista.PolyData

VTK polydata representation for PyVista visualization.

The representation lies in the X/Y plane within the standard global Cartesian coordinate system.

### Returns

```
pyvista.PolyData
```

VTK pyvista.Polydata configuration.

## **Description**

Provides for creating and managing a triangle.

### **Description**

PyAnsys Geometry sketch subpackage.

# The tools package

## **Summary**

### **Submodules**

check_geometry	Module for repair tool message.
measurement_tools	Provides tools for measurement.
prepare_tools	Provides tools for preparing geometry for use with simulation.
problem_areas	The problem area definition.
repair_tool_message	Module for repair tool message.
repair_tools	Provides tools for repairing bodies.
unsupported	Unsupported functions for the PyAnsys Geometry library.

# The check\_geometry.py module

# **Summary**

### Classes

GeometryIssue	Provides return message for the repair tool methods.
InspectResult	Provides the result of the inspect geometry operation.

## GeometryIssue

Provides return message for the repair tool methods.

## **Overview**

## **Properties**

message_type	The type of the message (warning, error, info).
message_id	The identifier for the message.
message	The content of the message.
edges	The List of edges (if any) that are part of the issue.
faces	The List of faces (if any) that are part of the issue.

# Import detail

from ansys.geometry.core.tools.check\_geometry import GeometryIssue

# **Property detail**

```
property GeometryIssue.message_type: str
    The type of the message (warning, error, info).
property GeometryIssue.message_id: str
    The identifier for the message.
```

```
property GeometryIssue.message: str
```

The content of the message.

### property GeometryIssue.edges: list[str]

The List of edges (if any) that are part of the issue.

### property GeometryIssue.faces: list[str]

The List of faces (if any) that are part of the issue.

### InspectResult

class ansys.geometry.core.tools.check\_geometry.InspectResult(grpc\_client: an-

sys.geometry.core.connection.client.GrpcClient,

body: an-

sys.geometry.core.designer.body.Body,

issues: list[GeometryIssue])

Provides the result of the inspect geometry operation.

#### Overview

### **Methods**

repair Repair the problem area.

## **Properties**

body	The body for which issues are found.
issues	The list of issues for the body.

### Import detail

from ansys.geometry.core.tools.check\_geometry import InspectResult

## **Property detail**

property InspectResult.body: ansys.geometry.core.designer.body.Body

The body for which issues are found.

property InspectResult.issues: list[GeometryIssue]

The list of issues for the body.

#### Method detail

 $\label{local_constraint} \textbf{InspectResult.repair}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ \textbf{Repair the problem area}.$ 

#### Returns

# ${\tt RepairToolMessage}$

Message containing created and/or modified bodies.

# **Description**

Module for repair tool message.

The measurement\_tools.py module

# **Summary**

### **Classes**

Gap	Represents a gap between two bodies.
${\it MeasurementTools}$	Measurement tools for PyAnsys Geometry.

#### Gap

**class** ansys.geometry.core.tools.measurement\_tools.**Gap**(*distance*: an-

sys.geometry.core.misc.measurements.Distance)

Represents a gap between two bodies.

#### **Parameters**

#### distance

[Distance] Distance between two sides of the gap.

### Overview

## **Properties**

distance	Returns the closest distance between two bodies.
----------	--------------------------------------------------

## Import detail

from ansys.geometry.core.tools.measurement\_tools import Gap

## **Property detail**

property Gap.distance: ansys.geometry.core.misc.measurements.Distance

Returns the closest distance between two bodies.

### MeasurementTools

class ansys.geometry.core.tools.measurement\_tools.MeasurementTools(grpc\_client: an-

sys.geometry.core.connection.GrpcClient)

Measurement tools for PyAnsys Geometry.

#### **Parameters**

# grpc\_client

[GrpcClient] gRPC client to use for the measurement tools.

### **Overview**

#### **Methods**

min\_distance\_between\_objects Find the gap between two bodies.

### Import detail

```
from ansys.geometry.core.tools.measurement_tools import MeasurementTools
```

#### **Method detail**

```
MeasurementTools.min_distance_between_objects(object1: ansys.geometry.core.designer.body.Body | ansys.geometry.core.designer.face.Face | ansys.geometry.core.designer.edge.Edge, object2: ansys.geometry.core.designer.body.Body | ansys.geometry.core.designer.face.Face | ansys.geometry.core.designer.face.Face | ansys.geometry.core.designer.edge.Edge) \rightarrow Gap
```

Find the gap between two bodies.

Gap between two bodies.

```
Parameters
```

```
object1
    [Union[Body, Face, Edge]] First object to measure the gap.
object2
    [Union[Body, Face, Edge]] Second object to measure the gap.
Returns
Gap
```

## **Description**

Provides tools for measurement.

The prepare\_tools.py module

# **Summary**

Classes

*PrepareTools* Prepare tools for PyAnsys Geometry.

### PrepareTools

```
\textbf{class} \ \ \textbf{ansys.geometry.core.tools.prepare\_tools.PrepareTools} (\textit{grpc\_client: ansys.geometry.core.connection.GrpcClient})
```

Prepare tools for PyAnsys Geometry.

### **Parameters**

```
grpc_client
```

[GrpcClient] Active supporting geometry service instance for design modeling.

### **Overview**

#### **Methods**

extract_volume_from_faces	Extract a volume from input faces.
<pre>extract_volume_from_edge_loops</pre>	Extract a volume from input edge loops.
share_topology	Share topology between the chosen bodies.
<pre>enhanced_share_topology</pre>	Share topology between the chosen bodies.

## Import detail

```
from ansys.geometry.core.tools.prepare_tools import PrepareTools
```

### **Method detail**

PrepareTools.extract\_volume\_from\_faces(sealing\_faces: list[ansys.geometry.core.designer.face.Face], inside\_faces: list[ansys.geometry.core.designer.face.Face])  $\rightarrow$  list[ansys.geometry.core.designer.body.Body]

Extract a volume from input faces.

Creates a volume (typically a flow volume) from a list of faces that seal the volume and one or more faces that define the wetted surface (inside faces of the solid).

#### **Parameters**

### sealing\_faces

[list[Face]] List of faces that seal the volume.

### inside\_faces

[list[Face]] List of faces that define the interior of the solid.

#### Returns

#### list[Body]

List of created bodies.

PrepareTools.extract\_volume\_from\_edge\_loops(sealing\_edges:

list[ansys.geometry.core.designer.edge.Edge],
inside\_faces: list[ansys.geometry.core.designer.face.Face]
= None) → list[ansys.geometry.core.designer.body.Body]

Extract a volume from input edge loops.

Creates a volume (typically a flow volume) from a list of edge loops that seal the volume. and one or more faces that define the wetted surface (inside faces of the solid).

#### **Parameters**

#### sealing\_edges

[list[Edge]] List of faces that seal the volume.

## inside\_faces

[list[Face], optional] List of faces that define the interior of the solid (not always necessary).

#### Returns

## list[Body]

List of created bodies.

```
PrepareTools.share_topology(bodies: list[ansys.geometry.core.designer.body.Body], tol:
                                   ansys.geometry.core.typing.Real = 0.0, preserve_instances: bool = False) \rightarrow
     Share topology between the chosen bodies.
           Parameters
               bodies
                   [list[Body]] List of bodies to share topology between.
               tol
                   [Real] Maximum distance between bodies.
               preserve_instances
                   [bool] Whether instances are preserved.
           Returns
               bool
                   True if successful, False if failed.
PrepareTools.enhanced_share_topology(bodies: list[ansys.geometry.core.designer.body.Body], tol:
                                              ansys.geometry.core.typing.Real = 0.0, preserve_instances: bool =
                                              False) \rightarrow
                                              ansys.geometry.core.tools.repair_tool_message.RepairToolMessage
     Share topology between the chosen bodies.
           Parameters
               bodies
                   [list[Body]] List of bodies to share topology between.
               tol
                   [Real] Maximum distance between bodies.
               preserve_instances
                   [bool] Whether instances are preserved.
           Returns
               RepairToolMessage
                   Message containing number of problem areas found/fixed, created and/or modified bodies.
Description
Provides tools for preparing geometry for use with simulation.
The problem_areas.py module
```

**Summary** 

## Classes

ProblemArea	Represents problem areas.
DuplicateFaceProblemAreas	Provides duplicate face problem area definition.
MissingFaceProblemAreas	Provides missing face problem area definition.
InexactEdgeProblemAreas	Represents an inexact edge problem area with unique identifier and associated edges.
ExtraEdgeProblemAreas	Represents a extra edge problem area with unique identifier and associated edges.
ShortEdgeProblemAreas	Represents a short edge problem area with a unique identifier and associated edges.
SmallFaceProblemAreas	Represents a small face problem area with a unique identifier and associated faces.
SplitEdgeProblemAreas	Represents a split edge problem area with unique identifier and associated edges.
StitchFaceProblemAreas	Represents a stitch face problem area with unique identifier and associated faces.
UnsimplifiedFaceProblemAre	Represents a unsimplified face problem area with unique identifier and associated faces.
InterferenceProblemAreas	Represents an interference problem area with a unique identifier and associated bodies.

#### **ProblemArea**

Represents problem areas.

# **Parameters**

id

[str] Server-defined ID for the problem area.

grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

# **Overview**

## **Abstract methods**

*fix* Fix problem area.

# **Properties**

id The id of the problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import ProblemArea

### **Property detail**

```
property ProblemArea.id: str
```

The id of the problem area.

#### **Method detail**

#### abstractmethod ProblemArea.fix()

Fix problem area.

### DuplicateFaceProblemAreas

class ansys.geometry.core.tools.problem\_areas.DuplicateFaceProblemAreas(id: str, grpc\_client:

an.

sys.geometry.core.connection.GrpcClient faces:

list[ansys.geometry.core.designer.face.Fa

Bases: ProblemArea

Provides duplicate face problem area definition.

Represents a duplicate face problem area with unique identifier and associated faces.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

# grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### faces

[list[Face]] List of faces associated with the design.

#### Overview

#### **Methods**

fix Fix the problem area.

# **Properties**

faces The list of faces connected to this problem area.

## Import detail

from ansys.geometry.core.tools.problem\_areas import DuplicateFaceProblemAreas

## **Property detail**

property DuplicateFaceProblemAreas.faces: list[ansys.geometry.core.designer.face.Face]

The list of faces connected to this problem area.

 $\label{eq:def-def-def-def} \mbox{DuplicateFaceProblemAreas.} \textbf{fix}() \rightarrow \mbox{ansys.geometry.core.tools.repair\_tool\_message.} RepairToolMessage \\ \mbox{Fix the problem area.}$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

#### MissingFaceProblemAreas

class ansys.geometry.core.tools.problem\_areas.MissingFaceProblemAreas(id: str, grpc\_client: an-

sys.geometry.core.connection.GrpcClient, edges:

list[ansys.geometry.core.designer.edge.Edge

Bases: ProblemArea

Provides missing face problem area definition.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### edges

[list[Edge]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

edges The list of edges connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import MissingFaceProblemAreas

## **Property detail**

 $\label{lem:missingFaceProblemAreas.fix} \textbf{MissingFaceProblemAreas.fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ \textbf{Fix the problem area.}$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

### InexactEdgeProblemAreas

class ansys.geometry.core.tools.problem\_areas.InexactEdgeProblemAreas(id: str, grpc\_client: an-

sys.geometry.core.connection.GrpcClient, edges:

list[ansys.geometry.core.designer.edge.Edge

Bases: ProblemArea

Represents an inexact edge problem area with unique identifier and associated edges.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### edges

[list[Edge]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

edges The list of edges connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import InexactEdgeProblemAreas

## **Property detail**

property InexactEdgeProblemAreas.edges: list[ansys.geometry.core.designer.edge.Edge]
 The list of edges connected to this problem area.

 $\label{lem:core.tools.repair_tool_message.RepairToolMessage} In exact \texttt{EdgeProblemAreas.fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ Fix the problem area.$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

### **ExtraEdgeProblemAreas**

class ansys.geometry.core.tools.problem\_areas.ExtraEdgeProblemAreas(id: str, grpc\_client: an-

sys.geometry.core.connection.GrpcClient,

edges:

list[ansys.geometry.core.designer.edge.Edge])

Bases: ProblemArea

Represents a extra edge problem area with unique identifier and associated edges.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### edges

[list[Edge]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

edges The list of edges connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import ExtraEdgeProblemAreas

## **Property detail**

property ExtraEdgeProblemAreas.edges: list[ansys.geometry.core.designer.edge.Edge]

The list of edges connected to this problem area.

 $\label{eq:core.tools.repair_tool_message.RepairToolMessage} \textbf{Fix the problem area.} \ \ \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage}$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

## ShortEdgeProblemAreas

class ansys.geometry.core.tools.problem\_areas.ShortEdgeProblemAreas(id: str, grpc\_client: an-

sys. geometry. core. connection. Grpc Client,

edges:

list[ansys.geometry.core.designer.edge.Edge])

Bases: ProblemArea

Represents a short edge problem area with a unique identifier and associated edges.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### edges

[list[Edge]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

edges The list of edges connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import ShortEdgeProblemAreas

## **Property detail**

property ShortEdgeProblemAreas.edges: list[ansys.geometry.core.designer.edge.Edge]
 The list of edges connected to this problem area.

 $\label{lem:core.tools.repair_tool_message.RepairToolMessage} Fix the problem area. \\ \textbf{fix()} \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ Fix the problem area. \\ \\ \textbf{fix()} \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ \textbf{fix()} \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.RepairToolMessage.Re$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

### SmallFaceProblemAreas

class ansys.geometry.core.tools.problem\_areas.SmallFaceProblemAreas(id: str, grpc\_client: an-

sys. geometry. core. connection. Grpc Client,

faces:

list[ansys.geometry.core.designer.face.Face])

Bases: ProblemArea

Represents a small face problem area with a unique identifier and associated faces.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### faces

[list[Face]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

faces The list of faces connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import SmallFaceProblemAreas

## **Property detail**

property SmallFaceProblemAreas.faces: list[ansys.geometry.core.designer.face.Face]
 The list of faces connected to this problem area.

 $\label{eq:smallFaceProblemAreas.fix} SmallFaceProblemAreas. \textbf{fix}() \rightarrow ansys. geometry. core. tools. repair\_tool\_message. RepairToolMessage \\ Fix the problem area.$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

### SplitEdgeProblemAreas

class ansys.geometry.core.tools.problem\_areas.SplitEdgeProblemAreas(id: str, grpc\_client: an-

sys. geometry. core. connection. Grpc Client,

edges:

list[ansys.geometry.core.designer.edge.Edge])

Bases: ProblemArea

Represents a split edge problem area with unique identifier and associated edges.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### edges

[list[Edge]] List of edges associated with the design.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

edges The list of edges connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import SplitEdgeProblemAreas

## **Property detail**

property SplitEdgeProblemAreas.edges: list[ansys.geometry.core.designer.edge.Edge]
 The list of edges connected to this problem area.

 $\label{eq:split} SplitEdgeProblemAreas. \textbf{fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message}. RepairToolMessage \\ Fix the problem area.$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

#### StitchFaceProblemAreas

class ansys.geometry.core.tools.problem\_areas.StitchFaceProblemAreas(id: str, grpc\_client: an-

sys.geometry.core.connection.GrpcClient,

bodies:

list[ansys.geometry.core.designer.body.Body

Bases: ProblemArea

Represents a stitch face problem area with unique identifier and associated faces.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### **bodies**

[list[Body]] List of bodies associated with the design.

## **Overview**

**Methods** 

fix Fix the problem area.

### **Properties**

bodies The list of bodies connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import StitchFaceProblemAreas

## **Property detail**

property StitchFaceProblemAreas.bodies: list[ansys.geometry.core.designer.body.Body]
The list of bodies connected to this problem area.

 $\label{eq:stitch} \textbf{StitchFaceProblemAreas.fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage} \\ \textbf{Fix the problem area.}$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

### UnsimplifiedFaceProblemAreas

class ansys.geometry.core.tools.problem\_areas.UnsimplifiedFaceProblemAreas(id: str,

grpc\_client: an-

sys. geometry. core. connection. Grpc C

faces:

list[ansys.geometry.core.designer.fac

Bases: ProblemArea

Represents a unsimplified face problem area with unique identifier and associated faces.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

#### grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### faces

[list[Face]] List of faces associated with the design.

#### Overview

### Methods

fix Fix the problem area.

## **Properties**

faces The list of faces connected to this problem area.

### Import detail

from ansys.geometry.core.tools.problem\_areas import UnsimplifiedFaceProblemAreas

### **Property detail**

property UnsimplifiedFaceProblemAreas.faces: list[ansys.geometry.core.designer.face.Face]
 The list of faces connected to this problem area.

 $\label{lem:condition} Unsimplified Face Problem Areas. \textbf{fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message}. Repair Tool Message Fix the problem area.$ 

#### **Returns**

### message: RepairToolMessage

Message containing created and/or modified bodies.

#### InterferenceProblemAreas

class ansys.geometry.core.tools.problem\_areas.InterferenceProblemAreas(id: str, grpc\_client: an-

sys.geometry.core.connection.GrpcClient, bodies:

list[ansys.geometry.core.designer.body.Bo

Bases: ProblemArea

Represents an interference problem area with a unique identifier and associated bodies.

#### **Parameters**

id

[str] Server-defined ID for the problem area.

## grpc\_client

[GrpcClient] Active supporting geometry service instance for design modeling.

#### **bodies**

[list[Body]] List of bodies in the problem area.

## **Overview**

### **Methods**

fix Fix the problem area.

### **Properties**

bodies The list of the bodies connected to this problem area.

# Import detail

from ansys.geometry.core.tools.problem\_areas import InterferenceProblemAreas

## **Property detail**

 $\label{lem:condition} \textbf{InterferenceProblemAreas.fix}() \rightarrow \textit{ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage}$  Fix the problem area.

#### **Returns**

## message: RepairToolMessage

Message containing created and/or modified bodies.

#### **Notes**

The current implementation does not properly track changes. The list of created and modified bodies are empty.

## **Description**

The problem area definition.

The repair\_tool\_message.py module

### **Summary**

#### **Classes**

RepairToolMessage Provides return message for the repair tool methods.

### RepairToolMessage

Provides return message for the repair tool methods.

#### Overview

## **Properties**

success	The success of the repair operation.
created_bodies	The list of the created bodies after the repair operation.
modified_bodies	The list of the modified bodies after the repair operation.
found	Number of problem areas found for the repair operation.
repaired	Number of problem areas repaired during the repair operation.

## Import detail

from ansys.geometry.core.tools.repair\_tool\_message import RepairToolMessage

## **Property detail**

property RepairToolMessage.success: bool

The success of the repair operation.

property RepairToolMessage.created\_bodies: list[str]

The list of the created bodies after the repair operation.

property RepairToolMessage.modified\_bodies: list[str]

The list of the modified bodies after the repair operation.

property RepairToolMessage.found: int

Number of problem areas found for the repair operation.

property RepairToolMessage.repaired: int

Number of problem areas repaired during the repair operation.

### **Description**

Module for repair tool message.

The repair\_tools.py module

**Summary** 

**Classes** 

RepairTools Repair tools for PyAnsys Geometry.

### **RepairTools**

class ansys.geometry.core.tools.repair\_tools.RepairTools(grpc\_client: an-

sys.geometry.core.connection.GrpcClient, modeler:

ansys.geometry.core.modeler.Modeler)

Repair tools for PyAnsys Geometry.

### Overview

### **Methods**

find_split_edges	Find split edges in the given list of bodies.
find_extra_edges	Find the extra edges in the given list of bodies.
find_inexact_edges	Find inexact edges in the given list of bodies.
find_short_edges	Find the short edge problem areas.
<pre>find_duplicate_faces</pre>	Find the duplicate face problem areas.
find_missing_faces	Find the missing faces.
find_small_faces	Find the small face problem areas.
find_stitch_faces	Return the list of stitch face problem areas.
find_simplify	Detect faces in a body that can be simplified.
find_interferences	Find the interference problem areas.
<pre>find_and_fix_short_edges</pre>	Find and fix the short edge problem areas.
<pre>find_and_fix_extra_edges</pre>	Find and fix the extra edge problem areas.
<pre>find_and_fix_split_edges</pre>	Find and fix the split edge problem areas.
<pre>find_and_fix_simplify</pre>	Find and simplify the provided geometry.
inspect_geometry	Return a list of geometry issues organized by body.
repair_geometry	Attempt to repair the geometry for the given bodies.

# Import detail

```
from ansys.geometry.core.tools.repair_tools import RepairTools
```

#### Method detail

```
RepairTools.find_split_edges(bodies: list[ansys.geometry.core.designer.body.Body], angle: 
 ansys.geometry.core.typing.Real = 0.0, length: 
 ansys.geometry.core.typing.Real = 0.0) \rightarrow 
 list[ansys.geometry.core.tools.problem_areas.SplitEdgeProblemAreas]
```

Find split edges in the given list of bodies.

This method finds the split edge problem areas and returns a list of split edge problem areas objects.

### **Parameters**

```
bodies
```

[list[Body]] List of bodies that split edges are investigated on.

#### angle

[Real] The maximum angle between edges.

### length

[Real] The maximum length of the edges.

### Returns

### list[SplitEdgeProblemAreas]

List of objects representing split edge problem areas.

RepairTools.find\_extra\_edges(bodies: list[ansys.geometry.core.designer.body.Body])  $\rightarrow$  list[ansys.geometry.core.tools.problem\_areas.ExtraEdgeProblemAreas]

Find the extra edges in the given list of bodies.

This method find the extra edge problem areas and returns a list of extra edge problem areas objects.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that extra edges are investigated on.

#### Returns

### list[ExtraEdgeProblemArea]

List of objects representing extra edge problem areas.

RepairTools.find\_inexact\_edges(bodies: list[ansys.geometry.core.designer.body.Body])  $\rightarrow$  list[ansys.geometry.core.tools.problem areas.InexactEdgeProblemAreas]

Find inexact edges in the given list of bodies.

This method find the inexact edge problem areas and returns a list of inexact edge problem areas objects.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that inexact edges are investigated on.

#### Returns

## list[InExactEdgeProblemArea]

List of objects representing inexact edge problem areas.

RepairTools.find\_short\_edges(bodies: list[ansys.geometry.core.designer.body.Body], length:

ansys.geometry.core.typing.Real = 0.0)  $\rightarrow$ 

list[ansys.geometry.core.tools.problem\_areas.ShortEdgeProblemAreas]

Find the short edge problem areas.

This method finds the short edge problem areas and returns a list of these objects.

### **Parameters**

#### bodies

[list[Body]] List of bodies that short edges are investigated on.

### Returns

## list[ShortEdgeProblemAreas]

List of objects representing short edge problem areas.

RepairTools. find\_duplicate\_faces(bodies: list[ansys.geometry.core.designer.body.Body])  $\rightarrow$ 

 $list[ansys.geometry.core.tools.problem\_areas.DuplicateFaceProblemAreas]$ 

Find the duplicate face problem areas.

This method finds the duplicate face problem areas and returns a list of duplicate face problem areas objects.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that duplicate faces are investigated on.

#### Returns

### list[DuplicateFaceProblemAreas]

List of objects representing duplicate face problem areas.

RepairTools.find\_missing\_faces(bodies: list[ansys.geometry.core.designer.body.Body])  $\rightarrow$  list[ansys.geometry.core.tools.problem\_areas.MissingFaceProblemAreas]

Find the missing faces.

This method find the missing face problem areas and returns a list of missing face problem areas objects.

### **Parameters**

#### **bodies**

[list[Body]] List of bodies that missing faces are investigated on.

#### Returns

#### list[MissingFaceProblemAreas]

List of objects representing missing face problem areas.

RepairTools.find\_small\_faces(bodies: list[ansys.geometry.core.designer.body.Body])  $\rightarrow$  list[ansys.geometry.core.tools.problem areas.SmallFaceProblemAreas]

Find the small face problem areas.

This method finds and returns a list of ids of small face problem areas objects.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that small faces are investigated on.

#### Returns

## list[SmallFaceProblemAreas]

List of objects representing small face problem areas.

RepairTools.find\_stitch\_faces(bodies: list[ansys.geometry.core.designer.body.Body]) →

 $list [{\it ansys.geometry.core.tools.problem\_areas.StitchFaceProblemAreas}]$ 

Return the list of stitch face problem areas.

This method find the stitch face problem areas and returns a list of ids of stitch face problem areas objects.

### **Parameters**

#### **bodies**

[list[Body]] List of bodies that stitchable faces are investigated on.

#### Returns

## list[StitchFaceProblemAreas]

List of objects representing stitch face problem areas.

RepairTools.find\_simplify(bodies: list[ansys.geometry.core.designer.body.Body]) →

 $list[ansys.geometry.core.tools.problem\_areas.UnsimplifiedFaceProblemAreas]$ 

Detect faces in a body that can be simplified.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies to search.

#### Returns

## list[UnsimplifiedFaceProblemAreas]

List of objects representing unsimplified face problem areas.

RepairTools.find\_interferences(bodies: list[ansys.geometry.core.designer.body.Body], cut\_smaller\_body:

```
bool = False) \rightarrow
```

list[ansys.geometry.core.tools.problem\_areas.InterferenceProblemAreas]

Find the interference problem areas.

#### **Parameters**

#### bodies

[list[Body]] List of bodies that small faces are investigated on.

#### cut smaller body

[bool, optional] Whether to cut the smaller body if an intererference is found. By default, False.

#### Returns

### list[InterfenceProblemAreas]

List of objects representing interference problem areas.

#### **Notes**

This method finds and returns a list of ids of interference problem areas objects.

```
RepairTools.find_and_fix_short_edges(bodies: list[ansys.geometry.core.designer.body.Body], length: 
 ansys.geometry.core.typing.Real = 0.0, comprehensive\_result: bool = False) \rightarrow ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage
```

Find and fix the short edge problem areas.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that short edges are investigated on.

#### length

[Real, optional] The maximum length of the edges. By default, 0.0.

#### comprehensive\_result

[bool, optional] Whether to fix all problem areas individually. By default, False.

#### Returns

### RepairToolMessage

Message containing number of problem areas found/fixed, created and/or modified bodies.

# Notes

This method finds the short edges in the bodies and fixes them.

```
\label{list_equal} RepairTools. \textbf{find\_and\_fix\_extra\_edges} (bodies: list[ansys.geometry.core.designer.body.Body], \\ comprehensive\_result: bool = False) \rightarrow \\ ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage
```

Find and fix the extra edge problem areas.

### **Parameters**

#### **bodies**

[list[Body]] List of bodies that short edges are investigated on.

#### length

[Real] The maximum length of the edges.

### comprehensive\_result

[bool, optional] Whether to fix all problem areas individually. By default, False.

#### Returns

#### RepairToolMessage

Message containing number of problem areas found/fixed, created and/or modified bodies.

#### **Notes**

This method finds the extra edges in the bodies and fixes them.

```
RepairTools.find_and_fix_split_edges(bodies: list[ansys.geometry.core.designer.body.Body], angle: 
 ansys.geometry.core.typing.Real = 0.0, length: 
 ansys.geometry.core.typing.Real = 0.0, comprehensive_result: bool 
 = False) \rightarrow 
 ansys.geometry.core.tools.repair_tool_message.RepairToolMessage
```

Find and fix the split edge problem areas.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies that split edges are investigated on.

#### angle

[Real, optional] The maximum angle between edges. By default, 0.0.

#### length

[Real, optional] The maximum length of the edges. By default, 0.0.

### comprehensive result

[bool, optional] Whether to fix all problem areas individually. By default, False.

#### **Returns**

### RepairToolMessage

Message containing number of problem areas found/fixed, created and/or modified bodies.

#### **Notes**

This method finds the extra edges in the bodies and fixes them.

```
RepairTools.find_and_fix_simplify(bodies: list[ansys.geometry.core.designer.body.Body], comprehensive\_result:\ bool = False) \rightarrow \\ ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage
```

Find and simplify the provided geometry.

#### **Parameters**

#### **bodies**

[list[Body]] List of bodies to be simplified.

#### comprehensive\_result

[bool, optional] Whether to fix all problem areas individually. By default, False.

### Returns

#### RepairToolMessage

Message containing number of problem areas found/fixed, created and/or modified bodies.

#### **Notes**

This method simplifies the provided geometry.

```
RepairTools.inspect_geometry(bodies: list[ansys.geometry.core.designer.body.Body] = None) \rightarrow list[ansys.geometry.core.tools.check_geometry.InspectResult]
```

Return a list of geometry issues organized by body.

This method inspects the geometry and returns a list of the issues grouped by the body where they are found.

#### **Parameters**

# bodies

[list[Body]] List of bodies to inspect the geometry for. All bodies are inspected if the argument is not given.

#### Returns

### list[IssuesByBody]

List of objects representing geometry issues and the bodies where issues are found.

RepairTools.repair\_geometry(bodies: list[ansys.geometry.core.designer.body.Body] = None)  $\rightarrow$  ansys.geometry.core.tools.repair\_tool\_message.RepairToolMessage

Attempt to repair the geometry for the given bodies.

This method inspects the geometry for the given bodies and attempts to repair them.

### **Parameters**

## **bodies**

[list[Body]] List of bodies where to attempt to repair the geometry. All bodies are repaired if the argument is not given.

### **Returns**

## RepairToolMessage

Message containing success of the operation.

## **Description**

Provides tools for repairing bodies.

## The unsupported.py module

### **Summary**

### **Classes**

*UnsupportedCommands* Provides unsupported commands for PyAnsys Geometry.

### **Enums**

*PersistentIdType* Type of persistent id.

### UnsupportedCommands

class ansys.geometry.core.tools.unsupported.UnsupportedCommands(grpc\_client: an-

sys.geometry.core.connection.GrpcClient, modeler: an-

sys.geometry.core.modeler.Modeler)

Provides unsupported commands for PyAnsys Geometry.

## **Parameters**

#### grpc\_client

[GrpcClient] gRPC client to use for the geometry commands.

#### modeler

[Modeler] Modeler instance to use for the geometry commands.

### **Overview**

### **Methods**

set_export_id	Set the persistent id for the moniker.
<pre>get_body_occurrences_from_import_id</pre>	Get all body occurrences whose master has the given import id.
<pre>get_face_occurrences_from_import_id</pre>	Get all face occurrences whose master has the given import id.
<pre>get_edge_occurrences_from_import_id</pre>	Get all edge occurrences whose master has the given import id.

## Import detail

```
from ansys.geometry.core.tools.unsupported import UnsupportedCommands
```

#### Method detail

UnsupportedCommands.set\_export\_id(moniker: str,  $id_type$ : PersistentIdType, value: str)  $\rightarrow$  None Set the persistent id for the moniker.

```
Parameters
```

```
moniker
```

[str] Moniker to set the id for.

id\_type

[PersistentIdType] Type of id.

value

[str] Id to set.

UnsupportedCommands.get\_body\_occurrences\_from\_import\_id(import\_id: str, id\_type: PersistentIdType)

 $\rightarrow$ 

list[ansys.geometry.core.designer.body.Body]

Get all body occurrences whose master has the given import id.

### **Parameters**

```
import_id
```

[str] Persistent id

id\_type

[PersistentIdType] Type of id

#### **Returns**

#### list[Body]

List of body occurrences.

UnsupportedCommands.get\_face\_occurrences\_from\_import\_id(import\_id: str, id\_type: PersistentIdType)

 $\rightarrow$ 

list[ansys.geometry.core.designer.face.Face]

Get all face occurrences whose master has the given import id.

#### **Parameters**

```
import id
```

[str] Persistent id.

id\_type

[PersistentIdType] Type of id.

### Returns

```
list[Face]
```

List of face occurrences.

UnsupportedCommands.get\_edge\_occurrences\_from\_import\_id(import\_id: str, id\_type: PersistentIdType)

 $\rightarrow$ 

list[ansys.geometry.core.designer.edge.Edge]

Get all edge occurrences whose master has the given import id.

#### **Parameters**

```
import_id
    [str] Persistent id.
id type
```

[PersistentIdType] Type of id.

#### **Returns**

## list[Edge]

List of edge occurrences.

## PersistentIdType

class ansys.geometry.core.tools.unsupported.PersistentIdType(\*args, \*\*kwds)

Bases: enum.Enum

Type of persistent id.

### Overview

## **Attributes**

PNAME PRIME\_ID

# Import detail

from ansys.geometry.core.tools.unsupported import PersistentIdType

#### Attribute detail

```
PersistentIdType.PNAME = 1
PersistentIdType.PRIME_ID = 700
```

## **Description**

Unsupported functions for the PyAnsys Geometry library.

# **Description**

PyAnsys Geometry tools subpackage.

## The errors.py module

## **Summary**

# **Exceptions**

GeometryRuntimeError	Provides error message when Geometry service passes a runtime error.
${\it GeometryExitedError}$	Provides error message to raise when Geometry service has exited.

#### **Functions**

handler	Pass signal to the custom interrupt handler.
<pre>protect_grpc</pre>	Capture gRPC exceptions and raise a more succinct error message.

### **Constants**

SIGINT\_TRACKER

### GeometryRuntimeError

exception ansys.geometry.core.errors.GeometryRuntimeError

Bases: RuntimeError

Provides error message when Geometry service passes a runtime error.

## Import detail

from ansys.geometry.core.errors import GeometryRuntimeError

## GeometryExitedError

**exception** ansys.geometry.core.errors.**GeometryExitedError**(*msg='Geometry service has exited.'*)

Bases: RuntimeError

Provides error message to raise when Geometry service has exited.

#### **Parameters**

msg

[str, default: "Geometry service has exited."] Message to raise.

## Import detail

from ansys.geometry.core.errors import GeometryExitedError

# **Description**

Provides PyAnsys Geometry-specific errors.

## **Module detail**

# errors.handler(sig, frame)

Pass signal to the custom interrupt handler.

## errors.protect\_grpc(func)

Capture gRPC exceptions and raise a more succinct error message.

This method captures the KeyboardInterrupt exception to avoid segfaulting the Geometry service.

While this works some of the time, it does not work all of the time. For some reason, gRPC still captures SIGINT.

errors.SIGINT\_TRACKER = []

## The logger.py module

## **Summary**

## **Classes**

PyGeometryCustomAdapter	Keeps the reference to the Geometry service instance name dynamic.
PyGeometryPercentStyle	Provides a common messaging style.
PyGeometryFormatter	Provides a Formatter class for overwriting default format styles.
InstanceFilter	Ensures that the instance_name record always exists.
Logger	Provides the logger used for each PyAnsys Geometry session.

## **Functions**

addfile_handler	Add a file handler to the input.
add_stdout_handler	Add a standout handler to the logger.

### **Attributes**

string\_to\_loglevel

## **Constants**

LOG_LEVEL
FILE_NAME
DEBUG
INFO
WARN
ERROR
CRITICAL
STDOUT_MSG_FORMAT
FILE_MSG_FORMAT
DEFAULT_STDOUT_HEADER
DEFAULT_FILE_HEADER
NEW_SESSION_HEADER
LOG

## ${\bf PyGeometryCustomAdapter}$

class ansys.geometry.core.logger.PyGeometryCustomAdapter(logger, extra=None)

Bases: logging.LoggerAdapter

Keeps the reference to the Geometry service instance name dynamic.

If you use the standard approach, which is supplying *extra* input to the logger, you must input Geometry service instances each time you do a log.

Using adapters, you only need to specify the Geometry service instance that you are referring to once.

### Overview

### Methods

process	Process the logging message and keyword arguments passed in to
log_to_file	Add a file handler to the logger.
log_to_stdout	Add a standard output handler to the logger.
setLevel	Change the log level of the object and the attached handlers.

# Attributes

level
file\_handler
stdout\_handler
logger
std\_out\_handler

# Import detail

from ansys.geometry.core.logger import PyGeometryCustomAdapter

### **Attribute detail**

```
PyGeometryCustomAdapter.level = None
```

PyGeometryCustomAdapter.file\_handler = None

PyGeometryCustomAdapter.stdout\_handler = None

PyGeometryCustomAdapter.logger

PyGeometryCustomAdapter.std\_out\_handler

#### Method detail

PyGeometryCustomAdapter.process(msg, kwargs)

Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you'll only need to override this one method in a LoggerAdapter subclass for your specific needs.

PyGeometryCustomAdapter.log\_to\_file( $filename: str = FILE\_NAME, level: int = LOG\_LEVEL$ )

Add a file handler to the logger.

#### **Parameters**

#### filename

[str, default: "pyansys-geometry.log"] Name of the file to write log messages to.

#### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

PyGeometryCustomAdapter.log\_to\_stdout(level=LOG\_LEVEL)

Add a standard output handler to the logger.

#### **Parameters**

#### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

PyGeometryCustomAdapter.setLevel(level='DEBUG')

Change the log level of the object and the attached handlers.

#### **Parameters**

#### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

### **PyGeometryPercentStyle**

**class** ansys.geometry.core.logger.**PyGeometryPercentStyle**(fmt, \* (Keyword-only parameters separator (PEP 3102)), defaults=None)

Bases: logging.PercentStyle

Provides a common messaging style.

### Import detail

```
from ansys.geometry.core.logger import PyGeometryPercentStyle
```

### **PyGeometryFormatter**

```
class ansys.geometry.core.logger.PyGeometryFormatter(fmt=STDOUT\_MSG\_FORMAT, datefmt=None, style='\%', validate=True, defaults=None)
```

Bases: logging.Formatter

Provides a Formatter class for overwriting default format styles.

### Import detail

```
from ansys.geometry.core.logger import PyGeometryFormatter
```

#### InstanceFilter

```
class ansys.geometry.core.logger.InstanceFilter(name=")
```

Bases: logging.Filter

Ensures that the instance\_name record always exists.

#### Overview

### **Methods**

*filter* Ensure that the instance\_name attribute is always present.

### Import detail

```
from ansys.geometry.core.logger import InstanceFilter
```

### **Method detail**

InstanceFilter.filter(record)

Ensure that the instance\_name attribute is always present.

#### Logger

Provides the logger used for each PyAnsys Geometry session.

This class allows you to add handlers to the logger to output messages to a file or to the standard output (stdout).

### **Parameters**

#### level

[int, default: 10] Logging level to filter the message severity allowed in the logger. The default is 10, in which case the logging.DEBUG level is used.

#### to file

[bool, default: False] Whether to write log messages to a file.

#### to stdout

[bool, default: True] Whether to write log messages to the standard output.

### filename

[str, default: "pyansys-geometry.log"] Name of the file to write log log messages to.

### **Examples**

Demonstrate logger usage from the Modeler instance, which is automatically created when a Geometry service instance is created.

```
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler(loglevel="DEBUG")
>>> modeler._log.info("This is a useful message")
INFO - - <ipython-input-24-80df150fe31f> - <module> - This is LOG debug message.
```

Import the global PyAnsys Geometry logger and add a file output handler.

```
>>> import os
>>> from ansys.geometry.core import LOG
>>> file_path = os.path.join(os.getcwd(), "pyansys-geometry.log")
>>> LOG.log_to_file(file_path)
```

#### Overview

#### Methods

log_to_file	Add a file handler to the logger.
log_to_stdout	Add the standard output handler to the logger.
setLevel	Change the log level of the object and the attached handlers.
add_child_logger	Add a child logger to the main logger.
add_instance_logger	Add a logger for a Geometry service instance.
add_handling_uncaught_expections	Redirect the output of an exception to a logger.

#### **Attributes**

file_handler
std_out_handler
logger
level
debug
info
warning
error
critical
log

### **Special methods**

```
__getitem__ Overload the access method by item for the Logger class.
```

### Import detail

```
from ansys.geometry.core.logger import Logger
```

### Attribute detail

```
Logger.file_handler = None

Logger.std_out_handler = None

Logger.logger

Logger.level = 0

Logger.debug

Logger.info

Logger.warning

Logger.error

Logger.critical

Logger.log
```

### Method detail

```
Logger.log_to_file(filename=FILE_NAME, level=LOG_LEVEL)
```

Add a file handler to the logger.

#### **Parameters**

### filename

[str, default: "pyansys-geometry.log"] Name of the file to write log messages to.

### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

### **Examples**

Write to the "pyansys-geometry.log" file in the current working directory:

```
>>> from ansys.geometry.core import LOG
>>> import os
>>> file_path = os.path.join(os.getcwd(), "pyansys-geometry.log")
>>> LOG.log_to_file(file_path)
```

### Logger.log\_to\_stdout(level=LOG\_LEVEL)

Add the standard output handler to the logger.

#### **Parameters**

#### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

#### Logger.setLevel(level='DEBUG')

Change the log level of the object and the attached handlers.

```
Logger.add_child_logger(suffix: str, level: str | None = None)
```

Add a child logger to the main logger.

This logger is more general than an instance logger, which is designed to track the state of Geometry service instances.

If the logging level is in the arguments, a new logger with a reference to the \_global logger handlers is created instead of a child logger.

#### **Parameters**

#### suffix

[str] Name of the child logger.

#### level

[str, default: None] Level of logging.

#### Returns

### logging.Logger

Logger class.

Logger.add\_instance\_logger(name: str,  $client_instance$ : ansys.geometry.core.connection.client.GrpcClient, level:  $int \mid None = None$ )  $\rightarrow PyGeometryCustomAdapter$ 

Add a logger for a Geometry service instance.

The Geometry service instance logger is a logger with an adapter that adds contextual information such as the Geometry service instance name. This logger is returned, and you can use it to log events as a normal logger. It is stored in the \_instances field.

### **Parameters**

### name

[str] Name for the new instance logger.

#### client instance

[GrpcClient] Geometry service GrpcClient object, which should contain the get\_name method.

### level

[int, default: None] Level of logging.

#### Returns

### PyGeometryCustomAdapter

Logger adapter customized to add Geometry service information to the logs. You can use this class to log events in the same way you would with the Logger class.

#### Logger.\_\_getitem\_\_(key)

Overload the access method by item for the Logger class.

### Logger.add\_handling\_uncaught\_expections(logger)

Redirect the output of an exception to a logger.

#### **Parameters**

#### logger

[str] Name of the logger.

### **Description**

Provides a general framework for logging in PyAnsys Geometry.

This module is built on the Logging facility for Python. It is not intended to replace the standard Python logging library but rather provide a way to interact between its logging class and PyAnsys Geometry.

The loggers used in this module include the name of the instance, which is intended to be unique. This name is printed in all active outputs and is used to track the different Geometry service instances.

#### Logger usage

### Global logger

There is a global logger named PyAnsys\_Geometry\_global that is created when ansys.geometry.core. \_\_init\_\_ is called. If you want to use this global logger, you must call it at the top of your module:

```
from ansys.geometry.core import LOG
```

You can rename this logger to avoid conflicts with other loggers (if any):

```
from ansys.geometry.core import LOG as logger
```

The default logging level of LOG is ERROR. You can change this level and output lower-level messages with this code:

```
LOG.logger.setLevel("DEBUG")
LOG.file_handler.setLevel("DEBUG") # If present.
LOG.stdout_handler.setLevel("DEBUG") # If present.
```

Alternatively, you can ensure that all the handlers are set to the input log level with this code:

```
LOG.setLevel("DEBUG")
```

This logger does not log to a file by default. If you want, you can add a file handler with this code:

```
import os
file_path = os.path.join(os.getcwd(), "pyansys-geometry.log")
LOG.log_to_file(file_path)
```

This also sets the logger to be redirected to this file. If you want to change the characteristics of this global logger from the beginning of the execution, you must edit the \_\_init\_\_ file in the directory ansys.geometry.core.

To log using this logger, call the desired method as a normal logger with:

```
>>> import logging
>>> from ansys.geometry.core.logging import Logger
>>> LOG = Logger(level=logging.DEBUG, to_file=False, to_stdout=True)
>>> LOG.debug("This is LOG debug message.")
```

```
DEBUG - - <ipython-input-24-80df150fe31f> - <module> - This is LOG debug message.
```

### Instance logger

Every time an instance of the *Modeler* class is created, a logger is created and stored in LOG.\_instances. This field is a dictionary where the key is the name of the created logger.

These instance loggers inherit the PyAnsys\_Geometry\_global output handlers and logging level unless otherwise specified. The way this logger works is very similar to the global logger. If you want to add a file handler, you can use the  $log_to_file()$  method. If you want to change the log level, you can use the setLevel() method.

Here is an example of how you can use this logger:

```
>>> from ansys.geometry.core import Modeler
>>> modeler = Modeler()
>>> modeler._log.info("This is a useful message")

INFO - GRPC_127.0.0.1:50056 - <...> - <module> - This is a useful message
```

### Other loggers

You can create your own loggers using a Python logging library as you would do in any other script. There would be no conflicts between these loggers.

### Module detail

```
logger.addfile_handler(logger, filename=FILE_NAME, level=LOG_LEVEL, write_headers=False)
Add a file handler to the input.
```

#### **Parameters**

### logger

[logging.Logger] Logger to add the file handler to.

### filename

[str, default: "pyansys-geometry.log"] Name of the output file.

#### level

[int, default: 10] Level of logging. The default is 10, in which case the logging.DEBUG level is used.

#### write headers

[bool, default: False] Whether to write the headers to the file.

#### Returns

### Logger

Logger or logging.Logger object.

logger.add\_stdout\_handler(logger, level=LOG\_LEVEL, write\_headers=False)

Add a standout handler to the logger.

### **Parameters**

### logger

[logging.Logger] Logger to add the file handler to.

```
level
                [int, default: 10] Level of logging. The default is 10, in which case the logging. DEBUG
                level is used.
             write headers
                 [bool, default: False] Whether to write headers to the file.
         Returns
             Logger
                Logger or logging.Logger object.
logger.LOG_LEVEL = 10
logger.FILE_NAME = 'pyansys-geometry.log'
logger.DEBUG = 10
logger.INFO = 20
logger.WARN = 30
logger.ERROR = 40
logger.CRITICAL = 50
logger.STDOUT_MSG_FORMAT = '%(levelname)s - %(instance_name)s - %(module)s - %(funcName)s
- %(message)s'
logger.FILE_MSG_FORMAT = '%(levelname)s - %(instance_name)s - %(module)s - %(funcName)s -
%(message)s'
logger.DEFAULT_STDOUT_HEADER = Multiline-String
     LEVEL - INSTANCE NAME - MODULE - FUNCTION - MESSAGE
logger.DEFAULT_FILE_HEADER = Multiline-String
     LEVEL - INSTANCE NAME - MODULE - FUNCTION - MESSAGE
logger.NEW_SESSION_HEADER = Multiline-String
```

```
NEW SESSION - Uninferable
```

logger.LOG

logger.string\_to\_loglevel

### The modeler.py module

#### Summary

#### **Classes**

**Modeler** Provides for interacting with an open session of the Geometry service.

#### Modeler

```
class ansys.geometry.core.modeler.Modeler(host: str = DEFAULT\_HOST, port: str \mid int = DEFAULT\_PORT, channel: grpc.Channel \mid None = None, remote_instance: ansys.platform.instancemanagement.Instance \mid None = None, docker_instance: ansys.geometry.core.connection.docker_instance.LocalDockerInstance \mid None = None, product_instance: ansys.geometry.core.connection.product_instance.ProductInstance \mid None = None, timeout: ansys.geometry.core.typing.Real = 120, logging_level: int = logging.INFO, logging_file: pathlib.Path \mid str \mid None = None, backend_type: ansys.geometry.core.connection.backend.BackendType \mid None = None)
```

Provides for interacting with an open session of the Geometry service.

#### **Parameters**

#### host

[str, default: DEFAULT\_HOST] Host where the server is running.

## port

[str | int, default: DEFAULT\_PORT] Port number where the server is running.

#### channel

[Channel, default: None] gRPC channel for server communication.

### remote\_instance

[ansys.platform.instancemanagement.Instance, default: None] Corresponding remote instance when the Geometry service is launched using PyPIM. This instance is deleted when the GrpcClient.close method is called.

#### docker instance

[LocalDockerInstance, default: None] Corresponding local Docker instance when the Geometry service is launched using the launch\_docker\_modeler method. This instance is deleted when the GrpcClient.close method is called.

### product\_instance

default: prod-[ProductInstance. None Corresponding local net instance when the product (Discovery or SpaceClaim) launched the launch\_modeler\_with\_geometry\_service(), through launch\_modeler\_with\_discovery() or the launch\_modeler\_with\_spaceclaim() interface. This instance will be deleted when the GrpcClient.close method is called.

#### timeout

[Real, default: 120] Time in seconds for trying to achieve the connection.

### logging\_level

[int, default: INFO] Logging level to apply to the client.

### logging\_file

[str, Path, default: None] File to output the log to, if requested.

### **Overview**

### Methods

create_design	Initialize a new design with the connected client.
get_active_design	Get the active design on the modeler object.
read_existing_design	Read the existing design on the service with the connected client.
close	Access the client's close method.
exit	Access the client's close method.
open_file	Open a file.
<pre>run_discovery_script_file</pre>	Run a Discovery script file.
get_service_logs	Get the service logs.

## **Properties**

client	Modeler instance client.
design	Retrieve the design within the modeler workspace.
designs	Retrieve the design within the modeler workspace.
repair_tools	Access to repair tools.
prepare_tools	Access to prepare tools.
measurement_tools	Access to measurement tools.
<pre>geometry_commands</pre>	Access to geometry commands.
unsupported	Access to unsupported commands.

### **Special methods**

repr	Represent the modeler as a string.
repr	Represent the modeler as a string.

### Import detail

from ansys.geometry.core.modeler import Modeler

### **Property detail**

property Modeler.design: ansys.geometry.core.designer.design.Design
 Retrieve the design within the modeler workspace.

property Modeler.designs: dict[str, ansys.geometry.core.designer.design.Design]

Retrieve the design within the modeler workspace.

### **Notes**

This method is deprecated. Use the *design()* property instead.

property Modeler.repair\_tools: ansys.geometry.core.tools.repair\_tools.RepairTools
 Access to repair tools.

property Modeler.prepare\_tools: ansys.geometry.core.tools.prepare\_tools.PrepareTools
 Access to prepare tools.

property Modeler.measurement\_tools:

 $ansys. {\it geometry.core.tools.measurement\_tools.MeasurementTools}$ 

Access to measurement tools.

property Modeler.geometry\_commands:

ansys.geometry.core.designer.geometry\_commands.GeometryCommands

Access to geometry commands.

property Modeler.unsupported: ansys.geometry.core.tools.unsupported.UnsupportedCommands
Access to unsupported commands.

### **Method detail**

Modeler.create\_design(name: str)  $\rightarrow$  ansys.geometry.core.designer.design.Design Initialize a new design with the connected client.

#### **Parameters**

name

[str] Name for the new design.

#### Returns

Desian

Design object created on the server.

Modeler.get\_active\_design( $sync\_with\_backend: bool = True$ )  $\rightarrow ansys.geometry.core.designer.design.Design$ Get the active design on the modeler object.

#### **Parameters**

#### svnc with backend

[bool, default: True] Whether to sync the active design with the remote service. If set to False, the active design may be out-of-sync with the remote service. This is useful when the active design is known to be up-to-date.

#### Returns

Design

Design object already existing on the modeler.

 ${\tt Modeler.read\_existing\_design()} \rightarrow {\it ansys.geometry.core.designer.design.Design}$ 

Read the existing design on the service with the connected client.

### Returns

### Design

Design object already existing on the server.

```
Modeler.close(close\_design: bool = True) \rightarrow None
```

Access the client's close method.

#### **Parameters**

#### close design

[bool, default: True] Whether to close the design before closing the client.

```
Modeler.exit(close\_design: bool = True) \rightarrow None
```

Access the client's close method.

#### **Parameters**

### close\_design

[bool, default: True] Whether to close the design before closing the client.

#### **Notes**

This method is calling the same method as close().

```
Modeler.open_file(file_path: str \mid pathlib.Path, upload\_to\_server: bool = True, import\_options: ansys.geometry.core.misc.options.ImportOptions = ImportOptions()) \rightarrow ansys.geometry.core.designer.design.Design
```

Open a file.

This method imports a design into the service. On Windows, .scdocx and HOOPS Exchange formats are supported. On Linux, only the .scdocx format is supported.

If the file is a shattered assembly with external references, the whole containing folder will need to be uploaded. Ensure proper folder structure in order to prevent the uploading of unnecessary files.

#### **Parameters**

#### file\_path

[str, Path] Path of the file to open. The extension of the file must be included.

### upload\_to\_server

[bool] True if the service is running on a remote machine. If service is running on the local machine, set to False, as there is no reason to upload the file.

### import\_options

[ImportOptions] Import options that toggle certain features when opening a file.

#### Returns

#### Design

Newly imported design.

```
Modeler.__repr__() \rightarrow str
```

Represent the modeler as a string.

```
Modeler.run_discovery_script_file(file_path: str | pathlib.Path, script_args: dict[str, str] | None = None, import_design: bool = False, api_version: int | str | ansys.geometry.core.connection.backend.ApiVersions = None) → tuple[dict[str, str], ansys.geometry.core.designer.design.Design | None]
```

Run a Discovery script file.

#### 1 Note

If arguments are passed to the script, they must be in the form of a dictionary. On the server side, the script will receive the arguments as a dictionary of strings, under the variable name argsDict. For example, if the script is called with the arguments run\_discovery\_script\_file(..., script\_args = {"length": "20"}, ...), the script will receive the dictionary argsDict with the key-value pair {"length": "20"}.

### 1 Note

If an output is expected from the script, it will be returned as a dictionary of strings. The keys and values of the dictionary are the variables and their values that the script returns. However, it is necessary that the script creates a dictionary called result with the variables and their values that are expected to be returned. For example, if the script is expected to return the number of bodies in the design, the script should create a dictionary called result with the key-value pair {"numBodies": numBodies}, where numBodies is the number of bodies in the design.

The implied API version of the script should match the API version of the running Geometry Service. DMS API versions 24.1 and later are supported. DMS is a Windows-based modeling service that has been containerized to ease distribution, execution, and remotability operations.

#### **Parameters**

#### file\_path

[str | Path] Path of the file. The extension of the file must be included.

[dict[str, str], optional.] Arguments to pass to the script. By default, None.

### import\_design

[bool, optional.] Whether to refresh the current design from the service. When the script is expected to modify the existing design, set this to True to retrieve up-to-date design data. When this is set to False (default) and the script modifies the current design, the design may be out-of-sync. By default, False.

### api version

[int | str | ApiVersions, optional] The scripting API version to use. For example, version 24.1 can be passed as an integer 241, a string "241" or using the ansys.geometry. core.connection.backend.ApiVersions enum class. By default, None. When specified, the service will attempt to run the script with the specified API version. If the API version is not supported, the service will raise an error. If you are using Discovery or Space-Claim, the product will determine the API version to use, so there is no need to specify this parameter.

#### Returns

### dict[str, str]

Values returned from the script.

#### Design, optional

Up-to-date current design. This is only returned if import\_design=True.

#### Raises

#### GeometryRuntimeError

If the Discovery script fails to run. Otherwise, assume that the script ran successfully.

#### **Notes**

The Ansys Geometry Service only supports scripts that are of the same version as the running service. Any api\_version input will be ignored.

Modeler.get\_service\_logs( $all\_logs: bool = False, dump\_to\_file: bool = False, logs\_folder: str | pathlib.Path | None = None) <math>\rightarrow$  str | dict[str, str] | pathlib.Path

Get the service logs.

#### **Parameters**

#### all\_logs

[bool, default: False] Flag indicating whether all logs should be retrieved. By default, only the current logs are retrieved.

#### dump\_to\_file

[bool, default: False] Flag indicating whether the logs should be dumped to a file. By default, the logs are not dumped to a file.

### logs\_folder

[str, Path or None, default: None] Name of the folder where the logs should be dumped. This parameter is only used if the dump\_to\_file parameter is set to True.

#### Returns

#### str

Service logs as a string. This is returned if the dump\_to\_file parameter is set to False.

#### dict[str, str]

Dictionary containing the logs. The keys are the logs names, and the values are the logs as strings. This is returned if the all\_logs parameter is set to True and the dump\_to\_file parameter is set to False.

### Path

Path to the folder containing the logs (if the all\_logs parameter is set to True) or the path to the log file (if only the current logs are retrieved). The dump\_to\_file parameter must be set to True.

### **Description**

Provides for interacting with the Geometry service.

### The typing.py module

#### **Summary**

#### **Attributes**

Real	Type used to refer to both integers and floats as possible values.
RealSequence	Type used to refer to Real types as a Sequence type.

#### **Description**

Provides typing of values for PyAnsys Geometry.

### Module detail

### typing.Real

Type used to refer to both integers and floats as possible values.

### typing.RealSequence

Type used to refer to Real types as a Sequence type.

#### **Notes**

numpy.ndarrays are also accepted because they are the overlaying data structure behind most PyAnsys Geometry objects.

# 7.1.2 Description

PyAnsys Geometry is a Python wrapper for the Ansys Geometry service.

#### 7.1.3 Module detail

### core.USE\_SERVICE\_COLORS: bool = False

Global constant for checking whether to use service colors for plotting purposes. If set to False, the default colors will be used (speed gain).

### core.DISABLE\_MULTIPLE\_DESIGN\_CHECK: bool = True

Deprecated constant. Use DISABLE\_ACTIVE\_DESIGN\_CHECK instead.

### core.DISABLE\_ACTIVE\_DESIGN\_CHECK: bool = False

Global constant for disabling the ensure\_design\_is\_active check.

Only set this to false if you are sure you want to disable this check and your objects will always exist on the server side.

### core.DOCUMENTATION\_BUILD: bool

Global flag for the documentation to use the proper PyVista Jupyter backend.

### core.\_\_version\_\_

PyAnsys Geometry version.

**CHAPTER** 

**EIGHT** 

## **EXAMPLES**

These examples demonstrate the behavior and usage of PyAnsys Geometry.

# 8.1 PyAnsys Geometry 101 examples

These examples demonstrate basic operations you can perform with PyAnsys Geometry.

**1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.1.1 PyAnsys Geometry 101: Math

The math module is the foundation of PyAnsys Geometry. This module is built on top of NumPy, one of the most renowned mathematical Python libraries.

This example shows some of the main PyAnsys Geometry math objects and demonstrates why they are important prior to doing more exciting things in PyAnsys Geometry.

### **Perform required imports**

Perform the required imports.

[1]: import numpy as np

### Create points and vectors

Everything starts with Point and Vector objects, which can each be defined in a 2D or 3D form. These objects inherit from NumPy's ndarray, providing them with enhanced functionalities. When creating these objects, you must remember to pass in the arguments as a list (that is, with brackets [ ]).

Create 2D and 3D point and vectors.

```
Point3D([x, y, z])
Point2D([x, y])

(continues on next page)
```

```
Vector3D([x, y, z])
Vector2D([x, y])
```

You can perform standard mathematical operations on points and vectors.

Perform some standard operations on vectors.

```
[2]: vec_1 = Vector3D([1,0,0]) # x-vector
    vec_2 = Vector3D([0,1,0]) # y-vector

    print("Sum of vectors [1, 0, 0] + [0, 1, 0]:")
    print(vec_1 + vec_2) # sum

    print("\nDot product of vectors [1, 0, 0] * [0, 1, 0]:")
    print(vec_1 * vec_2) # dot

    print("\nCross product of vectors [1, 0, 0] % [0, 1, 0]:")
    print(vec_1 % vec_2) # cross

Sum of vectors [1, 0, 0] + [0, 1, 0]:
    [1 1 0]

    Dot product of vectors [1, 0, 0] * [0, 1, 0]:
    0

Cross product of vectors [1, 0, 0] % [0, 1, 0]:
    [0 0 1]
```

Create a vector from two points.

```
[3]: p1 = Point3D([12.4, 532.3, 89])
    p2 = Point3D([-5.7, -67.4, 46.6])

    vec_3 = Vector3D.from_points(p1, p2)
    vec_3

[3]: Vector3D([ -18.1, -599.7, -42.4])
```

Normalize a vector to create a unit vector, which is also known as a direction.

```
[4]: print("Magnitude of vec_3:")
    print("\nNormalized vec_3:")
    print(vec_3.normalize())

    print("\nNew magnitude:")
    print(vec_3.normalize().magnitude)

Magnitude of vec_3:
    601.4694173438911

Normalized vec_3:
    [-0.03009297 -0.99705818 -0.07049402]
```

```
New magnitude:
1.0
```

Use the UnitVector class to automatically normalize the input for the unit vector.

```
[5]: uv = UnitVector3D([1,1,1])
uv

[5]: UnitVector3D([0.57735027, 0.57735027])
```

Perform a few more mathematical operations on vectors.

```
[6]: v1 = Vector3D([1, 0, 0])
    v2 = Vector3D([0, 1, 0])
    print("Vectors are perpendicular:")
    print(v1.is_perpendicular_to(v2))
    print("\nVectors are parallel:")
    print(v1.is_parallel_to(v2))
    print("\nVectors are opposite:")
    print(v1.is_opposite(v2))
    print("\nAngle between vectors:")
    print(v1.get_angle_between(v2))
    print(f''{np.pi / 2} == pi/2")
    Vectors are perpendicular:
    True
    Vectors are parallel:
    False
    Vectors are opposite:
    False
    Angle between vectors:
    1.5707963267948966 radian
    1.5707963267948966 == pi/2
```

### **Create planes**

Once you begin creating sketches and bodies, Plane objects become very important. A plane is defined by these items:

- An origin, which consists of a 3D point
- Two directions (direction\_x and direction\_y), which are both UnitVector3Dobjects

If no direction vectors are provided, the plane defaults to the XY plane.

Create two planes.

```
[7]: plane = Plane(Point3D([0,0,0])) # XY plane

(continues on next page)
```

```
print("(1, 2, 0) is in XY plane:")
print(plane.is_point_contained(Point3D([1, 2, 0]))) # True

print("\n(0, 0, 5) is in XY plane:")
print(plane.is_point_contained(Point3D([0, 0, 5]))) # False

(1, 2, 0) is in XY plane:
True

(0, 0, 5) is in XY plane:
False
```

### Perform parametric evaluations

PyAnsys Geometry implements parametric evaluations for some curves and surfaces.

Evaluate a sphere.

```
[8]: from ansys.geometry.core.shapes import Sphere, SphereEvaluation
    from ansys.geometry.core.math import Point3D
    from ansys.geometry.core.misc import Distance

sphere = Sphere(Point3D([0,0,0]), Distance(1)) # radius = 1

eval = sphere.project_point(Point3D([1,1,1]))

print("U Parameter:")
    print(eval.parameter.u)

print("\nV Parameter:")
    print(eval.parameter.v)

U Parameter:
    0.7853981633974483

V Parameter:
    0.6154797086703873
```

```
[9]: print("Point on the sphere:")
  eval.position
  Point on the sphere:
[9]: Point3D([0.57735027, 0.57735027])
```

```
[10]: print("Normal to the surface of the sphere at the evaluation position:")
    eval.normal
    Normal to the surface of the sphere at the evaluation position:
[10]: UnitVector3D([0.57735027, 0.57735027])
```

### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

### Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.1.2 PyAnsys Geometry 101: Units

To handle units inside the source code, PyAnsys Geometry uses Pint, a third-party open source software that other PyAnsys libraries also use.

The following code examples show how to operate with units inside the PyAnsys Geometry codebase and create objects with different units.

### Import units handler

The following line of code imports the units handler: pint.util.UnitRegistry. For more information on the UnitRegistry class in the pint API, see Most important classes in the Pint documentation.

```
[1]: from ansys.geometry.core.misc import UNITS
```

### Create and work with Quantity objects

With the UnitRegistry object called UNITS, you can create Quantity objects. A Quantity object is simply a container class with two core elements:

- A number
- A unit

Quantity objects have convenience methods, including those for transforming to different units and comparing magnitudes, values, and units. For more information on the Quantity class in the pint API, see Most important classes in the Pint documentation. You can also step through this Pint tutorial.

```
[2]: from pint import Quantity
    a = Quantity(10, UNITS.mm)

print(f"Object a is a pint.Quantity: {a}")

print("Request its magnitude in different ways (accessor methods):")
print(f"Magnitude: {a.m}.")
print(f"Also magnitude: {a.magnitude}.")

print("Request its units in different ways (accessor methods):")
print(f"Units: {a.u}.")
print(f"Also units: {a.units}.")
```

```
# Quantities can be compared between different units
# You can also build Quantity objects as follows:
a2 = 10 * UNITS.mm
print(f''Compare quantities built differently: {a == a2}'')
# Quantities can be compared between different units
a2_diff_units = 1 * UNITS.cm
print(f"Compare quantities with different units: {a == a2_diff_units}")
Object a is a pint.Quantity: 10 millimeter
Request its magnitude in different ways (accessor methods):
Magnitude: 10.
Also magnitude: 10.
Request its units in different ways (accessor methods):
Units: millimeter.
Also units: millimeter.
Compare quantities built differently: True
Compare quantities with different units: True
```

PyAnsys Geometry objects work by returning Quantity objects whenever the property requested has a physical meaning.

Return Quantity objects for Point3D objects.

```
[3]: from ansys.geometry.core.math import Point3D
    point_a = Point3D([1,2,4])
    print("============= Point3D([1,2,4]) ===========")
    print(f"Point3D is a numpy.ndarray in SI units: {point_a}.")
    print(f"However, request each of the coordinates individually...\n")
    print(f"X Coordinate: {point_a.x}")
    print(f"Y Coordinate: {point_a.y}")
    print(f"Z Coordinate: {point_a.z}\n")
    # Now, store the information with different units...
    point_a_km = Point3D([1,2,4], unit=UNITS.km)
    print("========== Point3D([1,2,4], unit=UNITS.km) =========")
    print(f"Point3D is a numpy.ndarray in SI units: {point_a_km}.")
    print(f"However, request each of the coordinates individually...\n")
    print(f"X Coordinate: {point_a_km.x}")
    print(f"Y Coordinate: {point_a_km.y}")
    print(f"Z Coordinate: {point_a_km.z}\n")
    # These points, although they are in different units, can be added together.
    res = point_a + point_a_km
    print(f"numpy.ndarray: {res}")
    print(f"X Coordinate: {res.x}")
    print(f"Y Coordinate: {res.y}")
    print(f"Z Coordinate: {res.z}")
    Point3D is a numpy.ndarray in SI units: [1. 2. 4.].
```

#### Use default units

PyAnsys Geometry implements the concept of default units.

```
[4]: from ansys.geometry.core.misc import DEFAULT_UNITS

print("=== Default unit length ===")
print(DEFAULT_UNITS.LENGTH)

print(DEFAULT_UNITS.ANGLE)

=== Default unit length ===
meter
=== Default unit angle ===
radian
```

It is important to differentiate between *client-side* default units and *server-side* default units. You are able to control both of them.

Print the default server unit length.

```
[5]: print("=== Default server unit length ===")
print(DEFAULT_UNITS.SERVER_LENGTH)

=== Default server unit length ===
meter
```

Use default units.

```
[6]: from ansys.geometry.core.math import Point2D
from ansys.geometry.core.misc import DEFAULT_UNITS

DEFAULT_UNITS.LENGTH = UNITS.mm
```

```
point_2d_default_units = Point2D([3, 4])
print("This is a Point2D with default units")
print(f"X Coordinate: {point_2d_default_units.x}")
print(f"Y Coordinate: {point_2d_default_units.y}")
print(f"numpy.ndarray value: {point_2d_default_units}")

# Revert back to original default units
DEFAULT_UNITS.LENGTH = UNITS.m

This is a Point2D with default units
X Coordinate: 3 millimeter
Y Coordinate: 4 millimeter
numpy.ndarray value: [0.003 0.004]
```

PyAnsys Geometry has certain auxiliary classes implemented that provide proper unit checking when assigning values. Although they are basically intended for internal use of the library, you can define them for use.

```
[7]: from ansys.geometry.core.misc import Angle, Distance
```

Start with Distance. The main difference between a Quantity object (that is, from pint import Quantity) and a Distance is that there is an active check on the units passed (in case they are not the default ones). Here are some examples.

The next two code examples show how unreasonable operations raise errors.

```
[9]: try:
    radius.value = 3 * UNITS.degrees
except TypeError as err:
    print(f"Error raised: {err}")

Error raised: The pint.Unit provided as an input should be a [length] quantity.
```

```
[10]: try:
    radius.unit = UNITS.fahrenheit
except TypeError as err:
    print(f"Error raised: {err}")
```

Error raised: The pint.Unit provided as an input should be a [length] quantity.

The same behavior applies to the Angle object. Here are some examples.

```
[11]: import numpy as np

rotation_angle = Angle(np.pi / 2)
print(f"The rotation angle is {rotation_angle.value}.")

# Try reassigning the value of the distance
rotation_angle.value = 7 * UNITS.degrees
print(f"After reassignment, the rotation angle is {rotation_angle.value}.")

# You could also change its units if desired
rotation_angle.unit = UNITS.degrees
print(f"After changing its units, the rotation angle is {rotation_angle.value}.")

The rotation angle is 1.5707963267948966 radian.
After reassignment, the rotation angle is 0.12217304763960307 radian.
After changing its units, the rotation angle is 7.0 degree.
```

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

### 8.1.3 PyAnsys Geometry 101: Sketching

With PyAnsys Geometry, you can build powerful dynamic sketches without communicating with the Geometry service. This example shows how to build some simple sketches.

### **Perform required imports**

Perform the required imports.

```
[1]: from pint import Quantity

from ansys.geometry.core.math import Plane, Point2D, Point3D, Vector3D from ansys.geometry.core.misc import UNITS from ansys.geometry.core.sketch import Sketch
```

#### Add a box to sketch

The Sketch object is the starting point. Once it is created, you can dynamically add various curves to the sketch. Here are some of the curves that are available:

- arc
- box
- circle
- ellipse
- gear
- polygon
- segment
- slot
- trapezoid
- triangle

Add a box to the sketch.

A functional-style sketching API is also implemented. It allows you to append curves to the sketch with the idea of never picking up your pen.

Use the functional-style sketching API to add a box.

A Sketch object uses the XY plane by default. You can define your own custom plane using three parameters: origin, direction\_x, and direction\_y.

Add a box on a custom plane.

### Combine concepts to create powerful sketches

Combine these simple concepts to create powerful sketches.

```
[5]: # Complex Fluent API Sketch - PCB
    sketch = Sketch()
    (
        sketch.segment(Point2D([0, 0], unit=UNITS.mm), Point2D([40, 1], unit=UNITS.mm),
     →"LowerEdge")
           .arc_to_point(Point2D([41.5, 2.5], unit=UNITS.mm), Point2D([40, 2.5], unit=UNITS.
     →mm), tag="SupportedCorner")
           .segment_to_point(Point2D([41.5, 5], unit=UNITS.mm))
           .arc_to_point(Point2D([43, 6.5], unit=UNITS.mm), Point2D([43, 5], unit=UNITS.mm),
     →True)
           .segment_to_point(Point2D([55, 6.5], unit=UNITS.mm))
           .arc_to_point(Point2D([56.5, 8], unit=UNITS.mm), Point2D([55, 8], unit=UNITS.mm))
           .segment_to_point(Point2D([56.5, 35], unit=UNITS.mm))
           .arc_to_point(Point2D([55, 36.5], unit=UNITS.mm), Point2D([55, 35], unit=UNITS.mm))
          .segment_to_point(Point2D([0, 36.5], unit=UNITS.mm))
           .segment_to_point(Point2D([0, 0], unit=UNITS.mm))
           .circle(Point2D([4, 4], UNITS.mm), Quantity(1.5, UNITS.mm), "Anchor1")
           .circle(Point2D([51, 34.5], UNITS.mm), Quantity(1.5, UNITS.mm), "Anchor2")
    )
    sketch.plot()
    EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n <head>\n
                                                                                   <meta...
     →http-equiv="Content-...
```

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.1.4 PyAnsys Geometry 101: Modeling

Once you understand PyAnsys Geometry's mathematical constructs, units, and sketching capabilities, you can dive into its modeling capabilities.

PyAnsys Geometry is a Python client that connects to a modeling service. Here are the modeling services that are available for connection:

- **DMS**: Windows-based modeling service that has been containerized to ease distribution, execution, and remotability operations.
- Geometry service: Linux-based approach of DMS that is currently under development.
- Ansys Discovery and SpaceClaim: PyAnsys Geometry is capable of connecting to a running session of Ansys Discovery or SpaceClaim. Although this is not the main use case for PyAnsys Geometry, a connection to one of these Ansys products is possible. Because these products have graphical user interfaces, performance is not as high with this option as with the previous options. However, connecting to a running instance of Discovery or SpaceClaim might be useful for some users.

### Launch a modeling service

While the PyAnsys Geometry operations in earlier examples did not require communication with a modeling service, this example requires that a modeling service is available. All subsequent examples also require that a modeling service is available.

Launch a modeling service session.

```
[1]: from ansys.geometry.core import launch_modeler

# Start a modeler session
modeler = launch_modeler()
print(modeler)

Ansys Geometry Modeler (0x19a8bbfb320)

Ansys Geometry Modeler Client (0x19a8bd753a0)
    Target: localhost:700
    Connection: Healthy
```

You can also launch your own services and connect to them. For information on connecting to an existing service, see the Modeler API documentation.

Here is how the class architecture is implemented:

- Modeler: Handler object for the active service session. This object allows you to connect to an existing service by passing in a host and a port. It also allows you to create Design objects, which is where the modeling takes place. For more information, see the Modeler API documentation.
- Design: Root object of your assembly (tree). While a Design object is also a Component object, it has enhanced capabilities, including creating named selections, adding materials, and handling beam profiles. For more information, see the Design API documentation.
- Component: One of the main objects for modeling purposes. Component objects allow you to create bodies, subcomponents, beams, design points, planar surfaces, and more. For more information, see the Component API documentation.

The following code examples show how you use these objects. More capabilities of these objects are shown in the specific example sections for sketching and modeling.

### Create and plot a sketch

Create a Sketch object and plot it.

```
[2]: from ansys.geometry.core.sketch import Sketch
    from ansys.geometry.core.math import Point2D
    from ansys.geometry.core.misc import UNITS, Distance
    outer_hole_radius = Distance(0.5, UNITS.m)
    sketch = Sketch()
        sketch.segment(start=Point2D([-4, 5], unit=UNITS.m), end=Point2D([4, 5], unit=UNITS.
     →m))
        .segment_to_point(end=Point2D([4, -5], unit=UNITS.m))
        .segment_to_point(end=Point2D([-4, -5], unit=UNITS.m))
        .segment_to_point(end=Point2D([-4, 5], unit=UNITS.m))
        .box(
            center=Point2D([0, 0], unit=UNITS.m),
            width=Distance(3, UNITS.m),
            height=Distance(3, UNITS.m),
        )
        .circle(center=Point2D([3, 4], unit=UNITS.m), radius=outer_hole_radius)
        .circle(center=Point2D([-3, -4], unit=UNITS.m), radius=outer_hole_radius)
        .circle(center=Point2D([-3, 4], unit=UNITS.m), radius=outer_hole_radius)
        .circle(center=Point2D([3, -4], unit=UNITS.m), radius=outer_hole_radius)
    )
    # Plot the sketch
    sketch.plot()
    EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n <html>\n <head>\n
                                                                                    <meta_
     →http-equiv="Content-...
```

#### Perform some modeling operations

Now that the sketch is ready to be extruded, perform some modeling operations, including creating the design, creating the body directly on the design, and plotting the body.

### Perform some operations on the body

Perform some operations on the body.

```
[4]: # Request its faces, edges, volume...
faces = body.faces
edges = body.edges
volume = body.volume

print(f"This is body {body.name} with ID (server-side): {body.id}.")
print(f"This body has {len(faces)} faces and {len(edges)} edges.")
print(f"The body volume is {volume}.")

This is body Design_Body with ID (server-side): 0:22.
This body has 14 faces and 32 edges.
The body volume is 54.28672587712814 meter ** 3.
```

Other operations that can be performed include adding a midsurface offset and thickness (only for planar bodies), imprinting curves, assigning materials, copying, and translating.

Copy the body on a new subcomponent and translate it.

```
[5]: from ansys.geometry.core.math import UNITVECTOR3D_X

# Create a component
comp = design.add_component("Component")

# Copy the body that belongs to this new component
body_copy = body.copy(parent=comp, name="Design_Component_Body")

# Displace this new body by a certain distance (10m) in a certain direction (X-axis)
body_copy.translate(direction=UNITVECTOR3D_X, distance=Distance(10, unit=UNITS.m))

# Plot the result of the entire design
design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

Create and assign materials to the bodies that were created.

```
from pint import Quantity

from ansys.geometry.core.materials import Material, MaterialProperty,

MaterialPropertyType

# Define some general properties for the material.

density = Quantity(125, 10 * UNITS.kg / (UNITS.m * UNITS.m * UNITS.m))

poisson_ratio = Quantity(0.33, UNITS.dimensionless)

tensile_strength = Quantity(45) # WARNING: If no units are defined,

#it is assumed that the magnitude is in the units expected by the server.

# Once your material properties are defined, you can easily create a material.

material = Material(

"steel",

density,
```

Currently materials do not have any impact on the visualization when plotting is requested, although this could be a future feature. If the final assembly is open in Discovery or SpaceClaim, you can observe the changes.

#### Create a named selection

PyAnsys Geometry supports the creation of a named selection via the Design object.

Create a named selection with some of the faces of the previous body and the body itself.

#### Perform deletions

Deletion operations for bodies, named selections, and components are possible, always from the scope expected. For example, if you attempted to delete the original body from a component that has no ownership over it (such as your comp object), the deletion would fail. If you attempted to perform this deletion from the design object, the deletion would succeed.

The next two code examples show how deletion works.

```
\label{localize} $$EmbeddableWidget(value='<iframe srcdoc=''<!DOCTYPE html>\n<html>\n <head>\n <meta\_ http-equiv=&quot;Content-...
```

### **Export files**

Once modeling operations are finalized, you can export files in different formats. For the formats supported by DMS, see the DesignFileFormat class in the Design module documentation.

Export files in SCDOCX and FMD formats.

```
import os
from pathlib import Path

from ansys.geometry.core.designer import DesignFileFormat

# Path to downloads directory
file_dir = Path(os.getcwd(), "downloads")
file_dir.mkdir(parents=True, exist_ok=True)

# Download the model in different formats
design.download(file_location=Path(file_dir, "ModelingDemo.scdocx"),
format=DesignFileFormat.SCDOCX)
design.download(file_location=Path(file_dir, "ModelingDemo.fmd"),
format=DesignFileFormat.FMD)
```

### **Close session**

When you finish interacting with your modeling service, you should close the active server session. This frees resources wherever the service is running.

Close the server session.

[11]: modeler.close()



If the server session already existed (that is, it was not launched by the current client session), you cannot use this method to close the server session. You must manually close the server session instead. This is a safeguard for user-spawned services.

# Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.1.5 PyAnsys Geometry 101: Plotter

This example provides an overview of PyAnsys Geometry's plotting capabilities, focusing on its plotter features. After reviewing the fundamental concepts of sketching and modeling in PyAnsys Geometry, it shows how to leverage these key plotting capabilities:

- **Multi-object plotting**: You can conveniently plot a list of elements, including objects created in both PyAnsys Geometry and PyVista libraries.
- **Interactive object selection**: You can interactively select PyAnsys Geometry objects within the scene. This enables efficient manipulation of these objects in subsequent scripting.

### **Perform required imports**

Perform the required imports.

```
[1]: from pint import Quantity import pyvista as pv

from ansys.geometry.core.math import Point2D from ansys.geometry.core.misc import UNITS from ansys.geometry.core.plotting import GeometryPlotter from ansys.geometry.core.sketch import Sketch
```

## Launch modeling service

Launch a modeling service session.

```
[2]: from ansys.geometry.core import launch_modeler

# Start a modeler session
modeler = launch_modeler()
print(modeler)

Ansys Geometry Modeler (0x2175173a4b0)

Ansys Geometry Modeler Client (0x2173eb5c800)
    Target: localhost:700
    Connection: Healthy
```

You can also launch your own services and connect to them. For information on connecting to an existing service, see the Modeler API documentation.

### Instantiate design and initialize object list

Instantiate a new design to work on and initialize a list of objects for plotting.

```
[3]: # init modeler
design = modeler.create_design("Multiplot")

plot_list = []
```

You are now ready to create some objects and use the plotter capabilities.

### Create a PyAnsys Geometry body cylinder

Use PyAnsys Geometry to create a body cylinder.

```
[4]: cylinder = Sketch()
  cylinder.circle(Point2D([10, 10], UNITS.m), 1.0)
  cylinder_body = design.extrude_sketch("JustACyl", cylinder, Quantity(10, UNITS.m))
  plot_list.append(cylinder_body)
```

### Create a PyAnsys Geometry arc sketch

Use PyAnsys Geometry to create an arc sketch.

```
[5]: sketch = Sketch()
    sketch.arc(
        Point2D([20, 20], UNITS.m),
        Point2D([20, -20], UNITS.m),
        Point2D([10, 0], UNITS.m),
        tag="Arc",
    )
    plot_list.append(sketch)
```

### Create a PyVista cylinder

Use PyVista to create a cylinder.

```
[6]: cyl = pv.Cylinder(radius=5, height=20, center=(-20, 10, 10))
    plot_list.append(cyl)
```

#### Create a PyVista multiblock

Use PyVista to create a multiblock with a sphere and a cube.

### Create a PyAnsys Geometry body box

Use PyAnsys Geometry to create a body box that is a cube.

```
[8]: box2 = Sketch()
box2.box(Point2D([-10, 20], UNITS.m), Quantity(10, UNITS.m), Quantity(10, UNITS.m))
box_body2 = design.extrude_sketch("JustABox", box2, Quantity(10, UNITS.m))
plot_list.append(box_body2)
```

### **Plot objects**

When plotting the created objects, you have several options.

You can simply plot one of the created objects.

You can plot the whole list of objects.

The Python visualizer is used by default. However, you can also use trame for visualization.

```
plotter = GeometryPlotter(use_trame=True)
plotter.show(plot_list)
```

### Clip objects

You can clip any object represented in the plotter by defining a Plane object that intersects the target object.

```
[11]: from ansys.geometry.core.math import Plane, Point3D
    pl = GeometryPlotter()

# Define PyAnsys Geometry box
box2 = Sketch()
box2.box(Point2D([-10, 20], UNITS.m), Quantity(10, UNITS.m), Quantity(10, UNITS.m))
box_body2 = design.extrude_sketch("JustABox", box2, Quantity(10, UNITS.m))

# Define plane to clip the box
origin = Point3D([-10., 20., 5.], UNITS.m)
plane = Plane(origin=origin, direction_x=[1, 1, 1], direction_y=[-1, 0, 1])

# Add the object with the clipping plane
pl.plot(box_body2, clipping_plane=plane)
pl.show()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

### Select objects interactively

PyAnsys Geometry's plotter supports interactive object selection within the scene. This enables you to pick objects for subsequent script manipulation.

```
[12]: plotter = GeometryPlotter(allow_picking=True)

# Plotter returns picked bodies
picked_list = plotter.show(plot_list)
print(picked_list)

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
None
```

It is also possible to enable picking directly for a specific design or component object alone. In the following cell, picking is enabled for the design object.

#### Render in different colors

You can render the objects in different colors automatically using PyVista's default color cycler. In order to do this, activate the multi\_colors=True option when calling the plot() method.

In the following cell you can create a new design and plot a prism and a cylinder in different colors.

```
[14]: design = modeler.create_design("MultiColors")

# Create a sketch of a box
sketch_box = Sketch().box(Point2D([0, 0], unit=UNITS.m), width=30 * UNITS.m, height=40 *__
_UNITS.m)

# Create a sketch of a circle (overlapping the box slightly)
sketch_circle = Sketch().circle(Point2D([20, 0], unit=UNITS.m), radius=3 * UNITS.m)

# Extrude both sketches to get a prism and a cylinder
design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)
design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)

# Design plotting
design.plot(multi_colors=True)

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
_http-equiv=&quot;Content-...</pre>
```

### **Close session**

When you finish interacting with your modeling service, you should close the active server session. This frees resources wherever the service is running.

Close the server session.

[15]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.2 Sketching examples

These examples demonstrate math operations on geometric objects and sketching capabilities, combined with server-based operations.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.2.1 Sketching: Basic usage

This example shows how to use basic PyAnsys Geometry sketching capabilities.

### **Perform required imports**

Perform the required imports.

```
[1]: from ansys.geometry.core.misc.units import UNITS as u from ansys.geometry.core.sketch import Sketch
```

### Create a sketch

Sketches are fundamental objects for drawing basic shapes like lines, segments, circles, ellipses, arcs, and polygons.

You create a Sketch instance by defining a drawing plane. To define a plane, you declare a point and two fundamental orthogonal directions.

[2]: from ansys.geometry.core.math import Plane, Point2D, Point3D

Define a plane for creating a sketch.

```
[3]: # Define the origin point of the plane
    origin = Point3D([1, 1, 1])

# Create a plane located in previous point with desired fundamental directions
    plane = Plane(
```

```
origin, direction_x=[1, 0, 0], direction_y=[0, -1, 1]
)
# Instantiate a new sketch object from previous plane
sketch = Sketch(plane)
```

#### **Draw shapes**

To draw different shapes in the sketch, you use draw methods.

#### Draw a circle

You draw a circle in a sketch by specifying the center and radius.

#### Draw an ellipse

You draw an ellipse in a sketch by specifying the center, major radius, and minor radius.

#### Draw a polygon

You draw a regular polygon by specifying the center, radius, and desired number of sides.

#### Draw an arc

You draw an arc of circumference by specifying the center, starting point, and ending point.

There are also additional ways to draw arcs, such as by specifying the start, center point, and angle.

Or by specifying the start, end point, and radius.

#### Draw a slot

You draw a slot by specifying the center, width, and height.

#### Draw a box

You draw a box by specifying the center, width, and height.

```
[11]: sketch.box(Point2D([2, 0]), 4, 5, tag="Box")
    sketch.select("Box")
    sketch.plot_selection()
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...
```

#### Draw a segment

You draw a segment by specifying the starting point and ending point.

#### Plot the sketch

The Plotter class provides capabilities for plotting different PyAnsys Geometry objects. PyAnsys Geometry uses PyVista as the visualization backend.

You use the plot\_sketch method to plot a sketch. This method accepts a Sketch instance and some extra arguments to further customize the visualization of the sketch. These arguments include showing the plane of the sketch and its frame.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.2.2 Sketching: Dynamic sketch plane

The sketch is a lightweight, two-dimensional modeler driven primarily by client-side execution.

At any point, the current state of a sketch can be used for operations such as extruding a body, projecting a profile, or imprinting curves.

The sketch is designed as an effective *functional-style* API with all operations receiving 2D configurations.

For easy reuse of sketches across different regions of your design, you can move a sketch around the global coordinate system by modifying the plane defining the current sketch location.

This example creates a multi-layer PCB from many extrusions of the same sketch, creating unique design bodies for each layer.

# **Perform required imports**

Perform the required imports.

```
from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import UNITVECTOR3D_Z, Point2D
from ansys.geometry.core.misc import UNITS
from ansys.geometry.core.sketch import Sketch
```

#### Define sketch profile

You can create, modify, and plot Sketch instances independent of supporting Geometry service instances.

To define the sketch profile for the PCB, you create a sketch outline of individual Segment and Arc objects with two circular through-hole attachment points added within the profile boundary to maintain a single, closed sketch face.

Create a single Sketch instance to use for multiple design operations.

```
[2]: sketch = Sketch()
     (
        sketch.segment(Point2D([0, 0], unit=UNITS.mm), Point2D([40, 1], unit=UNITS.mm),
     →"LowerEdge")
           .arc_to_point(Point2D([41.5, 2.5], unit=UNITS.mm), Point2D([40, 2.5], unit=UNITS.
     →mm), tag="SupportedCorner")
           .segment_to_point(Point2D([41.5, 5], unit=UNITS.mm))
           .arc_to_point(Point2D([43, 6.5], unit=UNITS.mm), Point2D([43, 5], unit=UNITS.mm),
     →True)
           .segment_to_point(Point2D([55, 6.5], unit=UNITS.mm))
           .arc_to_point(Point2D([56.5, 8], unit=UNITS.mm), Point2D([55, 8], unit=UNITS.mm))
           .segment_to_point(Point2D([56.5, 35], unit=UNITS.mm))
           .arc_to_point(Point2D([55, 36.5], unit=UNITS.mm), Point2D([55, 35], unit=UNITS.mm))
           .segment_to_point(Point2D([0, 36.5], unit=UNITS.mm))
           .segment_to_point(Point2D([0, 0], unit=UNITS.mm))
           .circle(Point2D([4, 4], UNITS.mm), Quantity(1.5, UNITS.mm), "Anchor1")
           .circle(Point2D([51, 34.5], UNITS.mm), Quantity(1.5, UNITS.mm), "Anchor2")
    )
    sketch.plot()
    EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n
                                                                                   <meta_
     →http-equiv="Content-...
```

#### Instantiate the modeler

Launch a modeling service and connect to it.

```
[3]: modeler = launch_modeler()
    print(modeler)

Ansys Geometry Modeler (0x1f2fbf73d10)

Ansys Geometry Modeler Client (0x1f2f125ef90)
    Target: localhost:700
    Connection: Healthy
```

### **Extrude multiple bodies**

Use the single sketch profile to extrude the board profile at multiple Z-offsets. Create a named selection from the resulting list of layer bodies.

Note that translating the sketch plane prior to extrusion is more effective (10 server calls) than creating a design body on the supporting server and then translating the body on the server (20 server calls).

```
[4]: design = modeler.create_design("ExtrudedBoardProfile")

layers = []
layer_thickness = Quantity(0.20, UNITS.mm)
for layer_index in range(10):
    layers.append(design.extrude_sketch(f"BoardLayer_{layer_index}", sketch, layer_
    thickness))
    sketch.translate_sketch_plane_by_distance(UNITVECTOR3D_Z, layer_thickness)

board_named_selection = design.create_named_selection("FullBoard", bodies=layers)
design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
    http-equiv=&quot;Content-...</pre>
```

#### Close the modeler

Close the modeler to release the resources.

```
[5]: modeler.close()
```

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.2.3 Sketching: Parametric sketching for gears

This example shows how to use gear sketching shapes from PyAnsys Geometry.

### Perform required imports and pre-sketching operations

Perform required imports and instantiate the Modeler instance and the basic elements that define a sketch.

```
[1]: from pint import Quantity
    from ansys.geometry.core import launch_modeler
    from ansys.geometry.core.math import Plane, Point2D, Point3D
    from ansys.geometry.core.misc import UNITS, Distance
    from ansys.geometry.core.sketch import Sketch
    from ansys.geometry.core.plotting import GeometryPlotter
    # Start a modeler session
    modeler = launch_modeler()
    print(modeler)
    # Define the origin point of the plane
    origin = Point3D([1, 1, 1])
    # Create a plane containing the previous point with desired fundamental directions
    plane = Plane(
        origin, direction_x=[1, 0, 0], direction_y=[0, -1, 1]
    Ansys Geometry Modeler (0x20748056ff0)
    Ansys Geometry Modeler Client (0x2075182eff0)
      Target:
                  localhost:700
      Connection: Healthy
```

### Sketch a dummy gear

DummyGear sketches are simple gears that have straight teeth. While they do not ensure actual physical functionality, they might be useful for some simple playground tests.

Instantiate a new Sketch object and then define and plot a dummy gear.

```
\label{localize} Embeddable Widget(value='<iframe srcdoc=''<!DOCTYPE html>\n<html>\n <head>\n <meta\_ +http-equiv=&quot;Content-...
```

After creating the sketch, extrudes it.

# Sketch a spur gear

SpurGear sketches are parametric CAD spur gears based on four parameters:

- origin: Center point location for the desired spur gear. The value must be a Point2D object.
- module: Ratio between the pitch circle diameter in millimeters and the number of teeth. This is a common parameter for spur gears. The value should be an integer or a float.
- pressure\_angle: Pressure angle expected for the teeth of the spur gear. This is also a common parameter for spur gears. The value must be a pint.Quantity object.
- n\_teeth: Number of teeth. The value must be an integer.

Instantiate a new Sketch object and then define and plot a spur gear.

After creating the sketch, extrude it.

#### Close the modeler

[6]: modeler.close()

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3 Modeling examples

These examples demonstrate service-based modeling operations.

# Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.1 Modeling: Single body with material assignment

In PyAnsys Geometry, a *body* represents solids or surfaces organized within the Design assembly. The current state of sketch, which is a client-side execution, can be used for the operations of the geometric design assembly.

The Geometry service provides data structures to create individual materials and their properties. These data structures are exposed through PyAnsys Geometry.

This example shows how to create a single body from a sketch by requesting its extrusion. It then shows how to assign a material to this body.

#### **Perform required imports**

Perform the required imports.

```
from pint import Quantity

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.materials import Material, MaterialProperty,

MaterialPropertyType
from ansys.geometry.core.math import UNITVECTOR3D_Z, Frame, Plane, Point2D, Point3D,

(continues on next page)
```

#### Create sketch

Create a Sketch instance and insert a circle with a radius of 10 millimeters in the default plane.

```
[2]: sketch = Sketch()
    sketch.circle(Point2D([10, 10], UNITS.mm), Quantity(10, UNITS.mm))

[2]: <ansys.geometry.core.sketch.sketch.Sketch at 0x1ec7f8f08f0>
```

#### Initiate design on server

Launch a modeling service session and initiate a design on the server.

```
[3]: # Start a modeler session
    modeler = launch_modeler()
    print(modeler)

design_name = "ExtrudeProfile"
    design = modeler.create_design(design_name)

Ansys Geometry Modeler (0x1ec7f9aade0)

Ansys Geometry Modeler Client (0x1ec6cd2c7d0)
    Target: localhost:700
    Connection: Healthy
```

#### Add materials to design

Add materials and their properties to the design. Material properties can be added when creating the Material object or after its creation. This code adds material properties after creating the Material object.

#### Extrude sketch to create body

Extrude the sketch to create the body and then assign a material to it.

#### Close session

When you finish interacting with your modeling service, you should close the active server session. This frees resources wherever the service is running.

Close the server session.

[6]: modeler.close()

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.2 Modeling: Rectangular plate with multiple bodies

You can create multiple bodies from a single sketch by extruding the same sketch in different planes.

The sketch is designed as an effective *functional-style* API with all operations receiving 2D configurations. For more information, see the *Sketch* user guide.

In this example, a box is located in the center of the plate, with the default origin of a sketch plane (origin at (0, 0, 0)). Four holes of equal radius are sketched at the corners of the plate. The plate is then extruded, leading to the generation of the requested body. The projection is at the center of the face. The default projection depth is through the entire part.

# **Perform required imports**

Perform the required imports.

```
[1]: import numpy as np
from pint import Quantity

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import Plane, Point3D, Point2D, UnitVector3D
from ansys.geometry.core.misc import UNITS
from ansys.geometry.core.sketch import Sketch
```

#### Define sketch profile

The sketch profile for the proposed design requires four segments that constitute the outer limits of the design, a box on the center, and a circle at its four corners.

You can use a single sketch instance for multiple design operations, including extruding a body, projecting a profile, and imprinting curves.

Define the sketch profle for the rectangular plate with multiple bodies.

```
[2]: sketch = Sketch()
  (sketch.segment(Point2D([-4, 5], unit=UNITS.m), Point2D([4, 5], unit=UNITS.m))
        .segment_to_point(Point2D([4, -5], unit=UNITS.m))
        .segment_to_point(Point2D([-4, -5], unit=UNITS.m))
        .segment_to_point(Point2D([-4, 5], unit=UNITS.m))
        .box(Point2D([0,0], unit=UNITS.m), Quantity(3, UNITS.m), Quantity(3, UNITS.m))
        .circle(Point2D([3, 4], unit=UNITS.m), Quantity(0.5, UNITS.m))
        .circle(Point2D([-3, -4], unit=UNITS.m), Quantity(0.5, UNITS.m))
        .circle(Point2D([-3, 4], unit=UNITS.m), Quantity(0.5, UNITS.m))
        .circle(Point2D([3, -4], unit=UNITS.m), Quantity(0.5, UNITS.m))
        .circle(Point2D([3, -4], unit=UNITS.m), Quantity(0.5, UNITS.m))
)
```

#### Extrude sketch to create design

Establish a server connection and use the single sketch profile to extrude the base component at the Z axis. Create a named selection from the resulting list of bodies. In only three server calls, the design extrudes the four segments with the desired thickness.

#### Add component with a planar surface

After creating a plate as a base component, you might want to add a component with a planar surface to it.

Create a sketch instance and then create a surface in the design with this sketch. For the sketch, it creates an ellipse, keeping the origin of the plane as its center.

# Extrude from face to create body

Extrude a face profile by a given distance to create a solid body. There are no modifications against the body containing the source face.

### Translate body within plane

Use the :func:translate() <ansys.geometry.core.designer.body.Body.translate> method to move the body in a specified direction by a given distance. You can also move a sketch around the global coordinate system. For more information, see the *Dynamic Sketch Plane* example.

#### Close the modeler

Close the modeler to free up resources and release the connection.

[7]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.3 Modeling: Extruded plate with cut operations

As seen in the *Rectangular plate with multiple bodies* example, you can create a complex sketch with holes and extrude it to create a body. However, you can also perform cut operations on the extruded body to achieve similar results.

# Perform required imports

Perform the required imports.

```
from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import Plane, Point3D, Point2D
from ansys.geometry.core.misc import UNITS
from ansys.geometry.core.sketch import Sketch
```

### Define sketch profile without holes

Create a sketch profile for the proposed design. The sketch is the same as the *Rectangular plate with multiple bodies* example, but without the holes.

These holes are created by performing cut operations on the extruded body in the next steps.

#### Define sketch profile for holes

Create a sketch profile for the holes in the proposed design. The holes are created by sketching circles at the four corners of the plate. First create a reference sketch for all the circles. This sketch is translated to the four corners of the plate.

```
[3]: sketch_hole = Sketch()
    sketch_hole.circle(Point2D([0, 0], unit=UNITS.m), Quantity(0.5, UNITS.m))

hole_centers = [
    Plane(Point3D([3, 4, 0], unit=UNITS.m)),
    Plane(Point3D([-3, 4, 0], unit=UNITS.m)),
    Plane(Point3D([-3, -4, 0], unit=UNITS.m)),
    Plane(Point3D([3, -4, 0], unit=UNITS.m)),
    Plane(Point3D([3, -4, 0], unit=UNITS.m)),
]
```

## Perform cut operations on the extruded body

Perform cut operations on the extruded body to create holes at the four corners of the plate.

#### Close the modeler

Close the modeler to free up resources and release the connection.

[5]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.4 Modeling: Tessellation of two bodies

This example shows how to create two stacked bodies and return the tessellation as two merged bodies.

# **Perform required imports**

Perform the required imports.

```
from pint import Quantity

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import Point2D, Point3D, Plane
from ansys.geometry.core.misc import UNITS
from ansys.geometry.core.sketch import Sketch
```

## Create design

Create the basic sketches to be tessellated and extrude the sketch in the required plane. For more information on creating a component and extruding a sketch in the design, see the *Rectangular plate with multiple bodies* example.

Here is a typical situation in which two bodies, with different sketch planes, merge each body into a single dataset. This effectively combines all the faces of each individual body into a single dataset without separating faces.

```
[2]: modeler = launch_modeler()
    sketch_1 = Sketch()
    box = sketch_1.box(
        Point2D([10, 10], unit=UNITS.m), width=Quantity(10, UNITS.m), height=Quantity(5,_
     \hookrightarrow UNITS.m)
    circle = sketch_1.circle(
        Point2D([0, 0], unit=UNITS.m), radius=Quantity(25, UNITS.m)
    )
    design = modeler.create_design("TessellationDesign")
    comp = design.add_component("TessellationComponent")
    body = comp.extrude_sketch("Body", sketch=sketch_1, distance=10 * UNITS.m)
    # Create the second body in a plane with a different origin
    sketch_2 = Sketch(Plane([0, 0, 10]))
    box = sketch_2.box(Point2D(
         [10, 10], unit=UNITS.m), width=Quantity(10, UNITS.m), height=Quantity(5, UNITS.m)
    )
    circle = sketch_2.circle(
        Point2D([0, 10], unit=UNITS.m), radius=Quantity(25, UNITS.m)
    )
    body = comp.extrude_sketch("Body", sketch=sketch_2, distance=10 * UNITS.m)
```

#### Tessellate component as two merged bodies

Tessellate the component and merge each body into a single dataset. This effectively combines all the faces of each individual body into a single dataset without separating faces.

```
[3]: dataset = comp.tessellate()
dataset

PolyData (0x2737f744ac0)

N Cells: 3280

N Points: 3300

N Strips: 0

X Bounds: -2.500e+01, 2.500e+01

Y Bounds: -2.500e+01, 3.500e+01

Z Bounds: 0.000e+00, 2.000e+01

N Arrays: 0
```

Single body tessellation is possible. In that case, users can request the body-level tessellation method to tessellate the body and merge all the faces into a single dataset.

## Plot design

Plot the design.

[5]: design.plot()

```
\label{localize} $$EmbeddableWidget(value='<iframe srcdoc=''<!DOCTYPE html>\n <head>\n <meta\_ http-equiv=&quot;Content-...
```

#### Close the modeler

Close the modeler.

[6]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.5 Modeling: Design organization

The Design instance creates a design project within the remote Geometry service to complete all CAD modeling against.

You can organize all solid and surface bodies in each design within a customizable component hierarchy. A component is simply an organization mechanism.

The top-level design node and each child component node can have one or more bodies assigned and one or more components assigned.

The API requires each component of the design hierarchy to be given a user-defined name.

There are several design operations that result in a body being created within a design. Executing each of these methods against a specific component instance explicitly specifies the node of the design tree to place the new body under.

## **Perform required imports**

Perform the required imports.

```
[1]: from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import UNITVECTOR3D_X, Point2D
from ansys.geometry.core.misc import UNITS, Distance
from ansys.geometry.core.sketch import Sketch
```

# Organize design

Extrude two sketches to create bodies. Assign the cylinder to the top-level design component. Assign the slot to the component nested one level beneath the top-level design component.

```
[2]: modeler = launch_modeler()
    design = modeler.create_design("DesignHierarchyExample")
    circle_sketch = Sketch()
    circle_sketch.circle(Point2D([10, 10], UNITS.mm), Distance(10, UNITS.mm))
    cylinder_body = design.extrude_sketch("10mmCylinder", circle_sketch, Distance(10, UNITS.
     \rightarrowmm))
    slot_sketch = Sketch()
    slot_sketch.slot(Point2D([40, 10], UNITS.mm), Distance(20, UNITS.mm), Distance(10, UNITS.
     \rightarrowmm))
    nested_component = design.add_component("NestedComponent")
    slot_body = nested_component.extrude_sketch("SlotExtrusion", slot_sketch, Distance(20, __
     →UNITS.mm))
    design.plot()
    EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n
                                                                                     <meta_
     →http-equiv="Content-...
```

### **Create nested component**

Create a component that is nested under the previously created component and then create another cylinder from the previously used sketch.

## Use surfaces from body to create additional bodies

You can use surfaces from any body across the entire design as references for creating additional bodies.

Extrude a cylinder from the surface body assigned to the child component.

#### Close the modeler

Close the modeler to release the resources.

```
[5]: modeler.close()
```

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.6 Modeling: Boolean operations

This example shows how to use Boolean operations for geometry manipulation.

#### **Perform required imports**

Perform the required imports.

```
[1]: from ansys.geometry.core import launch_modeler from ansys.geometry.core.designer import Body from ansys.geometry.core.math import Point2D from ansys.geometry.core.misc import UNITS from ansys.geometry.core.plotting import GeometryPlotter from ansys.geometry.core.sketch import Sketch
```

#### Launch local modeler

Launch the local modeler. If you are not familiar with how to launch the local modeler, see the "Launch a modeling service" section in the *PyAnsys Geometry 101: Modeling* example.

```
[2]: modeler = launch_modeler()
    print(modeler)

Ansys Geometry Modeler (0x27333136690)

Ansys Geometry Modeler Client (0x27339f1b530)
    Target: localhost:700
    Connection: Healthy
```

#### **Define bodies**

This section defines the bodies to use the Boolean operations on. First you create sketches of a box and a circle, and then you extrude these sketches to create 3D objects.

#### Create sketches

Create sketches of a box and a circle that serve as the basis for your bodies.

```
[3]: # Create a sketch of a box
sketch_box = Sketch().box(Point2D([0, 0], unit=UNITS.m), width=30 * UNITS.m, height=40 *_
UNITS.m)

# Create a sketch of a circle (overlapping the box slightly)
sketch_circle = Sketch().circle(Point2D([20, 0], unit=UNITS.m), radius=10 * UNITS.m)
```

#### **Extrude sketches**

After the sketches are created, extrude them to create 3D objects.

```
[4]: # Create a design
  design = modeler.create_design("example_design")

# Extrude both sketches to get a prism and a cylinder

prism = design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)

cylin = design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)
```

You must extrude the sketches each time that you perform an example operation. This is because performing a Boolean operation modifies the underlying design permanently. Thus, you no longer have two bodies. As shown in the Boolean operations themselves, whenever you pass in a body, it is consumed, and so it no longer exists. The remaining body (with the performed Boolean operation) is the one that performed the call to the method.

#### Select bodies

You can optionally select bodies in the plotter as described in the "Select objects interactively" section in the *PyAnsys Geometry 101: Plotter* example. As shown in this example, the plotter preserves the picking order, meaning that the output list is sorted according to the picking order.

```
pl = GeometryPlotter(allow_picking=True)
pl.plot(design.bodies)
pl.show()
bodies: list[Body] = GeometryPlotter(allow_picking=True).show(design.bodies)
```

Otherwise, you can select bodies from the design directly.

```
[5]: bodies = [design.bodies[0], design.bodies[1]]
```

# **Perform Boolean operations**

This section performs Boolean operations on the defined bodies using the PyAnsys Geometry library. It explores intersection, union, and subtraction operations.

#### Perform an intersection operation

To perform an intersection operation on the bodies, first set up the bodies.

```
[6]: # Create a design
design = modeler.create_design("intersection_design")

# Extrude both sketches to get a prism and a cylinder
prism = design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)
cylin = design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)
```

Perform the intersection and plot the results.

The final remaining body is the prism body because the cylin body has been consumed.

```
[8]: print(design.bodies)
    ansys.geometry.core.designer.Body 0x27339862750
      Name
                           : Prism
      Exists
                           : True
      Parent component
                          : intersection_design
      MasterBody
                         : 0:22
      Surface body
                           : False
      Color
                           : #D6F7D1
    ]
```

# Perform a union operation

To carry out a union operation on the bodies, first set up the bodies.

```
[9]: # Create a design
  design = modeler.create_design("union_design")

# Extrude both sketches to get a prism and a cylinder
  prism = design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)
  cylin = design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)
```

Perform the union and plot the results.

```
[10]: prism.unite(cylin)
    _ = GeometryPlotter().show(design.bodies)
```

```
\label{localize} Embeddable Widget(value='<iframe srcdoc=''<!DOCTYPE html>\n<html>\n <head>\n <meta\_ +http-equiv=&quot;Content-...
```

The final remaining body is the prism body because the cylin body has been consumed.

### Perform a subtraction operation

To perform a subtraction operation on the bodies, first set up the bodies.

```
[12]: # Create a design
  design = modeler.create_design("subtraction_design")

# Extrude both sketches to get a prism and a cylinder
  prism = design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)
  cylin = design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)
```

Perform the subtraction and plot the results.

The final remaining body is the prism body because the cylin body has been consumed.

If you perform this action inverting the order of the bodies (that is, cylin.subtract(prism), you can see the difference in the resulting shape of the body.

Chapter 8. Examples

```
# Extrude both sketches to get a prism and a cylinder
prism = design.extrude_sketch("Prism", sketch_box, 50 * UNITS.m)
cylin = design.extrude_sketch("Cylinder", sketch_circle, 50 * UNITS.m)

# Invert subtraction
cylin.subtract(prism)
_ = GeometryPlotter().show(design.bodies)

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
_ http-equiv=&quot;Content-...</pre>
```

In this case, the final remaining body is the cylin body because the prism body has been consumed.

# [16]: print(design.bodies)

```
[
ansys.geometry.core.designer.Body 0x2733a088e60
Name : Cylinder
Exists : True
Parent component : subtraction_design_inverted
MasterBody : 0:85
Surface body : False
Color : #D6F7D1
]
```

#### Close the modeler

Close the modeler to release the resources.

### [17]: modeler.close()

#### **Summary**

These Boolean operations provide powerful tools for creating complex geometries and combining or modifying existing shapes in meaningful ways.

Feel free to experiment with different shapes, sizes, and arrangements to further enhance your understanding of Boolean operations in PyAnsys Geometry and their applications.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

### 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.7 Modeling: Scale, map and mirror bodies

The purpose of this notebook is to demonstrate the map() and scale() functions and their usage for transforming bodies.

```
import numpy as np

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import (
   Frame,
   Plane,
   Point2D,
   Point3D,
   UNITVECTOR3D_X,
   UNITVECTOR3D_Y,
   UNITVECTOR3D_Z,
)
from ansys.geometry.core.sketch import Sketch
```

#### Initialize the modeler

```
[2]: # Initialize the modeler for this example notebook
    m = launch_modeler()
    print(m)

Ansys Geometry Modeler (0x23589078470)

Ansys Geometry Modeler Client (0x2358979e660)
    Target: localhost:700
    Connection: Healthy
```

#### Scale body

The scale() function is designed to modify the size of 3D bodies by a specified scale factor. This function is a important part of geometric transformations, allowing for the dynamic resizing of bodies.

# Usage of scale()

To use the scale() function, you call it on an instance of a geometry body, passing a single argument: the scale value. This value is a real number (Real) that determines the factor by which the body's size is changed.

```
body.scale(value)
```

# **Example: Making a cube**

The following code snippets show how to change the size of a cube using the scale() function in Body objects. The process involves initializing a sketch design for the cube, defining the shape parameters, and then performing a rescaling operation to generate the new shape.

## Initialize the cube sketch design

A new design sketch named "cube" is created.

```
[3]: design = m.create_design("cube")
```

# **Define cube parameters**

side\_length is set to 10 units, representing the side length of the cube.

```
[4]: # Cube parameters side_length = 10
```

# Create the profile cube

A square box is created centered on the origin using side\_length as the side length of the square.

### Create cube body

extrude\_sketch on the box\_sketch as the base sketch and create the 3D cube with distance being the side\_length.

# Making the cube twice as large

• Copy the original cube. Using scale() with a value of 2, double the side lengths of the cube, thereby making the body twice as large, and then offset it to view the difference.

## Halfing the size of the original cube

Copy the original cube. Using scale() with a value of 0.5 effectively halves the side lengths of the cube. Then, offset the new cube to view the difference.

Note: Because the size of the cube in the previous cell was doubled, using the 0.25 factor translates it to half the size of the original cube.

```
[8]: # Copy the original cube
halved = cube.copy(cube.parent_component, "halved_box")
# Half the size
halved.scale(0.5)
# Translate the copied cube in the x direction
halved.translate(UNITVECTOR3D_X, -25)

design.plot()
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

### Map body

The map() function enables the reorientation of 3D bodies by mapping them onto a new specified frame. This function is used for adjusting the orientation of geometric bodies within 3D space to match specific reference frames. With this function, you are able to effectively perform translation and rotation operations in a single method by specifying a new frame.

# Usage of map()

To use the map() function, invoke it on an instance of a geometry body with a single argument: the new frame to map the body to. The frame is a structure or object that defines the new orientation parameters for the body.

```
body.map(new_frame)
```

#### **Example: Creating an asymmetric cube**

The following code snippets show how to use the map() function to reframe a cube body in the Body object. The process involves initializing a sketch design for the custom body, extruding the profile by a distance, and then performing the mapping operation to rotate the shape.

#### Initialize the shape sketch design

A new design sketch named "asymmetric\_cube" is created.

```
[9]: # Initialize the sketch design
design = m.create_design("asymmetric_cube")
```

### Create an asymmetric sketch profile

Make a sketch profile that is basically a cube centered on the origin with a side length of 2 with a cutout.

# Create the asymmetric body

extrude\_sketch on the asymmetric\_profile as the base sketch, creating the 3D cube with a cutout, with the distance being 1.

### Apply map reframing

First make a copy of the shape and translate it in 3D space so that you can view them side by side. Then, apply the reframing to the copied shape.

Note: The following map uses the default x direction, but the y direction is swapped with the z direction, effectively rotating the original shape so that it is standing vertically.

```
[12]: # Copy the body
copied_body = body.copy(body.parent_component, "copied_body")
# Apply the reframing
copied_body.map(Frame(Point3D([0, 0, 0]), UNITVECTOR3D_X, UNITVECTOR3D_Z))
# Shift the new modified body in the plane in the negative y direction by 2 units
copied_body.translate(UNITVECTOR3D_Y, -2)

design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

### Mirror body

The mirror() function is designed to mirror the geometry of a body across a specified plane. This function plays a role in geometric transformations, enabling the reflection of bodies to create symmetrical designs.

## Usage of mirror()

To use the mirror() function, you call it on an instance of a geometry body, passing a single argument: the plane across which to mirror the body. This plane is represented by a Plane object, defining the axis of symmetry for the mirroring operation.

```
body.mirror(plane)
```

# **Example: Triangle body**

The following code snippets show how to use the mirror() function to reframe a cube body in the Body object. The process involves initializing a sketch design for the body profile, extruding the profile by a distance, and then performing the mirroring operation to reflect the shape over the specified axis.

### Initialize the shape sketch design

A new design sketch named "triangle" is created.

```
[13]: # Initialize the sketch design
design = m.create_design("triangle")
```

### **Define parameters**

point1: First vertex of the triangle. point2: Second vertex of the triangle. point3: Third vertex of the triangle.

```
[14]: point1 = Point2D([5, 0])
   point2 = Point2D([2.5, 2.5])
   point3 = Point2D([2.5, -2.5])
```

#### Create triangle sketch profile

Using point1, point2, and point3, define the vertices of the triangle profile using those three points and then create line segments connecting them.

```
[15]: # Draw the triangle sketch profile
    sketch = Sketch()
    sketch.segment(start=point1, end=point2)
    sketch.segment(start=point2, end=point3)
    sketch.segment(start=point3, end=point1)

    sketch.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
    _http-equiv=&quot;Content-...</pre>
```

### **Create triangular body**

Using the sketch profile created in the previous step, use the extrude\_sketch method to create a solid body with a depth of 1.

```
[16]: # Extrude the triangular body by a distance of 1
    triangle = design.extrude_sketch("triangle_body", sketch, 1)
    design.plot()
```

# Mirror the triangular body

First, make a copy of the triangular body. Then, using mirror(), you can mirror the copied body over the ZY plane.

# Closing the modeler

[18]: m.close()

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.8 Modeling: Sweep chain and sweep sketch

This example shows how use the sweep\_sketch() and sweep\_chain() functions to create more complex extrusion profiles. You use the sweep\_sketch() function with a closed sketch profile and the sweep\_chain() function for an open profile.

```
[1]: # Imports
import numpy as np

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import (
   Plane,
   Point2D,
   Point3D,
   UNITVECTOR3D_X,
```

(continues on next page)

```
UNITVECTOR3D_Y,
  UNITVECTOR3D_Z,
)
from ansys.geometry.core.shapes import Circle, Ellipse, Interval
from ansys.geometry.core.sketch import Sketch
```

#### Initialize the modeler

```
[2]: # Initialize the modeler for this example notebook
    m = launch_modeler()
    print(m)

Ansys Geometry Modeler (0x16a7e09ca10)

Ansys Geometry Modeler Client (0x16a125589b0)
    Target: localhost:700
    Connection: Healthy
```

### **Example: Creating a donut**

The following code snippets show how to use the sweep\_sketch() function to create a 3D donut shape in the Design object. The process involves initializing a sketch design for the donut, defining two circles for the profile and path, and then performing a sweep operation to generate the donut shape.

### Initialize the donut sketch design

A new design sketch named "donut" is created.

```
[3]: # Initialize the donut sketch design
design_sketch = m.create_design("donut")
```

### **Define circle parameters**

path\_radius is set to 5 units, representing the radius of the circular path that the profile circle sweeps along. profile\_radius is set to 2 units, representing the radius of the profile circle that sweeps along the path to create the donut body.

```
[4]: # Donut parameters
path_radius = 5
profile_radius = 2
```

### Create the profile circle

A circle is created on the XZ-plane centered at the coordinates (5, 0, 0) with a radius defined by profile\_radius. This circle serves as the profile or cross-sectional shape of the donut.

```
[5]: # Create the circular profile on the XZ plane centered at (5, 0, 0)
    # with a radius of 2
    plane_profile = Plane(direction_x=UNITVECTOR3D_X, direction_y=UNITVECTOR3D_Z)
    profile = Sketch(plane=plane_profile)
    profile.circle(Point2D([path_radius, 0]), profile_radius)
```

(continues on next page)

### Create the path circle

Another circle, representing the path along which the profile circle is swept, is created on the XY-plane centered at (0, 0, 0). The radius of this circle is defined by path\_radius.

```
[6]: # Create the circular path on the XY plane centered at (0, 0, 0) with radius 5 path = [Circle(Point3D([0, 0, 0]), path_radius).trim(Interval(0, 2 * np.pi))]
```

# Perform the sweep operation

The sweep operation uses the profile circle and sweeps it along the path circle to create the 3D body of the donut. The result is stored in the variable body.

#### **Example: Creating a bowl**

This code demonstrates the process of using the sweep\_chain() function to create a 3D model of a stretched bowl in the Design object. The model is generated by defining a quarter-ellipse as a profile and sweeping it along a circular path, creating a bowl shape with a stretched profile.

### Initialize the bowl design

A design chain named "bowl" is created to initiate the bowl design process.

```
[8]: # Initialize the bowl sketch design
design_chain = m.create_design("bowl")
```

# **Define parameters**

radius is set to 10 units, used for both the profile and path definitions.

```
[9]: # Define the radius parameter
radius = 10
```

# Create the profile shape

A quarter-ellipse profile is created with a major radius equal to 10 units and a minor radius equal to 5 units. The ellipse is defined in a 3D space with a specific orientation and then trimmed to a quarter using an interval from 0 to pi/2 radians. This profile shapes the bowl's side.

```
[10]: # Create quarter-ellipse profile with major radius = 10, minor radius = 5
profile = [
    Ellipse(
        Point3D([0, 0, radius / 2]),
        radius,
        radius / 2,
        reference=UNITVECTOR3D_X,
        axis=UNITVECTOR3D_Y,
    ).trim(Interval(0, np.pi / 2))
]
```

# Create the path

A circular path is created, positioned parallel to the XY-plane but shifted upwards by 5 units (half the major radius). The circle has a radius of 10 units and is trimmed to form a complete loop with an interval from 0 to 2pi radians. This path defines the sweeping trajectory for the profile to create the bowl.

```
[11]: # Create circle on the plane parallel to the XY-plane but moved up
# by 5 units with radius 10
path = [Circle(Point3D([0, 0, radius / 2]), radius).trim(Interval(0, 2 * np.pi))]
```

### Perform the sweep operation

The bowl body is generated by sweeping the quarter-ellipse profile along the circular path. The sweep operation molds the profile shape along the path to form the stretched bowl. The result of this operation is stored in the variable body.

#### Closing the modeler

```
[13]: # Close the modeler
m.close()
```

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.9 Modeling: Revolving a sketch

This example shows how to use the revolve\_sketch() method to revolve a sketch around an axis to create a 3D body. You can also specify the angle of revolution to create a partial body.

```
[1]: # Imports
from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import (
    Plane,
    Point2D,
    Point3D,
    UNITVECTOR3D_X,
    UNITVECTOR3D_Z,
)
from ansys.geometry.core.misc import UNITS, Angle
from ansys.geometry.core.sketch import Sketch
```

#### Initialize the modeler

```
[2]: # Initialize the modeler for this example notebook
    m = launch_modeler()
    print(m)

Ansys Geometry Modeler (0x13bc71e2420)

Ansys Geometry Modeler Client (0x13bc78c26c0)
    Target: localhost:700
    Connection: Healthy
```

# **Example: Creating a quarter of a donut**

The following code snippets show how to use the revolve\_sketch() function to create a quarter of a 3D donut. The process involves defining a quarter of a circle as a profile and then revolving it around the Z-axis to create a 3D body.

# Initialize the sketch design

Create a design sketch named quarter-donut.

```
[3]: # Initialize the donut sketch design
design = m.create_design("quarter-donut")
```

#### **Define circle parameters**

Set path\_radius, which represents the radius of the circular path that the profile circle sweeps along, to 5 units. Set profile\_radius, which represents the radius of the profile circle that sweeps along the path to create the donut body, to 2 units.

```
[4]: # Donut parameters
path_radius = 5
profile_radius = 2
```

## Create the profile circle

Create a circle on the XZ plane centered at the coordinates (5, 0, 0) and useprofile\_radius to define the radius. This circle serves as the profile or cross-sectional shape of the donut.

```
[5]: # Create the circular profile on the XZ-plane centered at (5, 0, 0)
# with a radius of 2
plane_profile = Plane(
    origin=Point3D([path_radius, 0, 0]),
    direction_x=UNITVECTOR3D_X,
    direction_y=UNITVECTOR3D_Z,
)
profile = Sketch(plane=plane_profile)
profile.circle(Point2D([0, 0]), profile_radius)

profile.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
    http-equiv=&quot;Content-...</pre>
```

# Perform the revolve operation

Revolve the profile circle around the Z axis to create a quarter of a donut body. Set the angle of revolution to 90 degrees in the default direction, which is counterclockwise.

```
[6]: # Revolve the profile around the Z axis and center in the absolute origin
  # for an angle of 90 degrees
  design.revolve_sketch(
    "quarter-donut-body",
    sketch=profile,
    axis=UNITVECTOR3D_Z,
    angle=Angle(90, unit=UNITS.degrees),
    rotation_origin=Point3D([0, 0, 0]),
)

design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
    http-equiv=&quot;Content-...</pre>
```

### Perform a revolve operation with a negative angle of revolution

You can use a negative angle of revolution to create a quarter of a donut in the opposite direction. The following code snippet shows how to create a quarter of a donut in the clockwise direction. The same profile circle is used, but the angle of revolution is set to -90 degrees.

```
[7]: # Initialize the donut sketch design
  design = m.create_design("quarter-donut-negative")

# Revolve the profile around the Z axis and center in the absolute origin
# for an angle of -90 degrees (clockwise)
design.revolve_sketch(
    "quarter-donut-body-negative",
    sketch=profile,
```

(continues on next page)

#### Close the modeler

```
[8]: # Close the modeler
m.close()
```

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.10 Modeling: Exporting designs

After creating a design, you typically want to bring it into a CAD tool for further development. This notebook demonstrates how to export a design to the various supported CAD formats.

#### Create a design

The code creates a simple design for demonstration purposes. The design consists of a set of rectangular pads with a circular hole in the center.

```
[1]: from ansys.geometry.core import launch_modeler
    from ansys.geometry.core.math import UNITVECTOR3D_X, UNITVECTOR3D_Y, Point2D
    from ansys.geometry.core.sketch import Sketch

# Instantiate the modeler
    modeler = launch_modeler()

# Create a design
    design = modeler.create_design("ExportDesignExample")

# Create a sketch
    sketch = Sketch()
```

(continues on next page)

```
# Create a simple rectangle
sketch.box(Point2D([0, 0]), 10, 5)
# Extrude the sketch and displace the resulting body
# to make a plane of rectangles
for x_step in [-60, -45, -30, -15, 0, 15, 30, 45, 60]:
    for y_step in [-40, -30, -20, -10, 0, 10, 20, 30, 40]:
        # Extrude the sketch
       body = design.extrude_sketch(f"Body_X_{x_step}_Y_{y_step}", sketch, 5)
        # Displace the body in the x and y directions
       body.translate(UNITVECTOR3D_X, x_step)
       body.translate(UNITVECTOR3D_Y, y_step)
# Plot the design
design.plot()
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n
                                                                               <meta_
→http-equiv="Content-...
```

### **Export the design**

You can export the design to various CAD formats. For the formats supported see the DesignFileFormat class, which is part of the the design module documentation.

Nonetheless, there are a set of convenience methods that you can use to export the design to the supported formats. The following code snippets demonstrate how to do it. You can decide whether to export the design to a file in a certain directory or in the current working directory.

#### Export to a file in the current working directory

# Export to a file in a certain directory

#### **Export to SCDOCX format**

### **Export to Parasolid text format**

# **Export to Parasolid binary format**

#### **Export to STEP format**

#### **Export to IGES format**

### **Export to FMD format**

## **Export to PMDB format**

#### Close the modeler

Close the modeler after exporting the design.

```
[11]: # Close the modeler
modeler.close()
```

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.11 Modeling: Visualization of the design tree on terminal

A user can visualize its model object tree easily by using the tree\_print() method available on the Design and Component objects. This method prints the tree structure of the model in the terminal.

#### **Perform required imports**

For the following example, you need to import these modules:

```
from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math.constants import UNITVECTOR3D_X, UNITVECTOR3D_Y
from ansys.geometry.core.math.point import Point2D, Point3D
from ansys.geometry.core.misc.units import UNITS
from ansys.geometry.core.sketch.sketch import Sketch
```

#### Create a design

The following code creates a simple design for demonstration purposes. The design consists of several cylinders extruded. The interesting part is visualizing the corresponding design tree.

```
[2]: # Create a modeler object
    modeler = launch_modeler()
    # Create your design on the server side
    design = modeler.create_design("TreePrintComponent")
    # Create a Sketch object and draw a circle (all client side)
    sketch = Sketch()
    sketch.circle(Point2D([-30, -30]), 10 * UNITS.m)
    distance = 30 * UNITS.m
       The following component hierarchy is made
    #
    #
                 |---> comp_1 ---|--> nested_1_comp_1 ---> nested_1_nested_1_comp_1
    #
                                |---> nested_2_comp_1
    # DESIGN --- | --- comp_2 ---- nested_1_comp_2
    #
                1---> comp_3
    # Now, only "comp_3", "nested_2_comp_1" and "nested_1_nested_1_comp_1"
    # has a body associated.
```

```
# Create the components
comp_1 = design.add_component("Component_1")
comp_2 = design.add_component("Component_2")
comp_3 = design.add_component("Component_3")
nested_1_comp_1 = comp_1.add_component("Nested_1_Component_1")
nested_1_nested_1_comp_1 = nested_1_comp_1.add_component("Nested_1_Nested_1_Component_1")
nested_2_comp_1 = comp_1.add_component("Nested_2_Component_1")
nested_1_comp_2 = comp_2.add_component("Nested_1_Component_2")
# Create the bodies
b1 = comp_3.extrude_sketch(name="comp_3_circle", sketch=sketch, distance=distance)
b2 = nested_2_comp_1.extrude_sketch(
   name="nested_2_comp_1_circle", sketch=sketch, distance=distance
b2.translate(UNITVECTOR3D_X, 50)
b3 = nested_1_nested_1_comp_1.extrude_sketch(
   name="nested_1_nested_1_comp_1_circle", sketch=sketch, distance=distance
b3.translate(UNITVECTOR3D_Y, 50)
# Create beams (in design)
circle_profile_1 = design.add_beam_circular_profile(
    "CircleProfile1", Quantity(10, UNITS.mm), Point3D([0, 0, 0]), UNITVECTOR3D_X,
→UNITVECTOR3D_Y
beam_1 = nested_1_comp_2.create_beam(
   Point3D([9, 99, 999], UNITS.mm), Point3D([8, 88, 888], UNITS.mm), circle_profile_1
design.plot()
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n
                                                                               <meta...
→http-equiv="Content-...
```

#### Visualize the design tree

Now, let's visualize the design tree using the tree\_print() method. Let's start by printing the tree structure of the design object with no extra arguments.

```
[3]: design.tree_print()

>>> Tree print view of component 'TreePrintComponent'

Location
------
Root component (Design)

Subtree
-----
(comp) TreePrintComponent
|---(comp) Component_1
```

```
: |---(comp) Nested_1_Component_1
: : |---(comp) Nested_1_Nested_1_Component_1
: : |---(body) nested_1_nested_1_comp_1_circle
: |---(comp) Nested_2_Component_1
: |---(body) nested_2_comp_1_circle
|---(comp) Component_2
: |---(comp) Nested_1_Component_2
: |---(beam) 0:215
|---(comp) Component_3
|---(body) comp_3_circle
```

#### Controlling the depth of the tree

The tree\_print() method accepts an optional argument depth to control the depth of the tree to be printed. The default value is None, which means the entire tree is printed.

```
[4]: design.tree_print(depth=1)

>>> Tree print view of component 'TreePrintComponent'

Location
------
Root component (Design)

Subtree
-----
(comp) TreePrintComponent
|---(comp) Component_1
|---(comp) Component_2
|---(comp) Component_3
```

In this case, only the first level of the tree is printed - that is, the three main components.

#### Excluding bodies, components, or beams

By default, the tree\_print() method prints all the bodies, components, and beams in the design tree. However, you can exclude any of these by setting the corresponding argument to False.

```
|---(comp) Component_2
: |---(comp) Nested_1_Component_2
|---(comp) Component_3
```

In this case, the bodies and beams are not be printed in the tree structure.

```
[6]: design.tree_print(consider_comps=False)

>>> Tree print view of component 'TreePrintComponent'

Location
-----
Root component (Design)

Subtree
-----
(comp) TreePrintComponent
```

In this case, the components are not be printed in the tree structure - leaving only the design object represented.

### Sorting the tree

By default, the tree structure is sorted by the way the components, bodies, and beams were created. However, you can sort the tree structure by setting the sort\_keys argument to True. In that case, the tree is sorted alphabetically.

Let's add a new component to the design and print the tree structure by default.

```
[7]: comp_4 = design.add_component("A_Component")
    design.tree_print(depth=1)

>>> Tree print view of component 'TreePrintComponent'

Location
------
Root component (Design)

Subtree
-----
(comp) TreePrintComponent
|---(comp) Component_1
|---(comp) Component_2
|---(comp) Component_3
|---(comp) A_Component
```

Now, let's print the tree structure with the components sorted alphabetically.

```
[8]: design.tree_print(depth=1, sort_keys=True)

>>> Tree print view of component 'TreePrintComponent'

Location
------
Root component (Design)

Subtree
```

```
(comp) TreePrintComponent
|---(comp) A_Component
|---(comp) Component_1
|---(comp) Component_2
|---(comp) Component_3
```

#### Indenting the tree

By default, the tree structure is printed with an indentation level of 4. However, you can indent the tree structure by setting the indent argument to the desired value.

```
[9]: design.tree_print(depth=1, indent=8)

>>> Tree print view of component 'TreePrintComponent'

Location
------
Root component (Design)

Subtree
-----
(comp) TreePrintComponent
|-----(comp) Component_1
|-----(comp) Component_2
|-----(comp) Component_3
|-----(comp) A_Component
```

In this case, the tree structure is printed with an indentation level of 8.

#### Printing the tree from a specific component

You can print the tree structure from a specific component by calling the tree\_print() method on the component object.

```
[10]: nested_1_comp_1.tree_print()

>>> Tree print view of component 'Nested_1_Component_1'

Location
-----
TreePrintComponent > Component_1 > Nested_1_Component_1

Subtree
-----
(comp) Nested_1_Component_1
|---(comp) Nested_1_Nested_1_Component_1
|---(body) nested_1_nested_1_comp_1_circle
```

#### Closing the modeler

Finally, close the modeler.

[11]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.12 Modeling: Body color assignment and usage

In PyAnsys Geometry, a *body* represents solids or surfaces organized within the Design assembly. As users might be already familiar with, Ansys CAD products (like SpaceClaim, Ansys Discovery and the Geometry Service), allow to assign colors to bodies. This example shows how to assign colors to a body, retrieve their value and how to use them in the client-side visualization.

#### **Perform required imports**

Perform the required imports.

```
[1]: import ansys.geometry.core as pyansys_geometry

from ansys.geometry.core import launch_modeler
from ansys.geometry.core.math import Point2D, UNITVECTOR3D_X, UNITVECTOR3D_Y
from ansys.geometry.core.sketch import Sketch
```

#### Create a box sketch

Create a Sketch instance and insert a box sketch with a width and height of 10 in the default plane.

```
[2]: sketch = Sketch()
    sketch.box(Point2D([0, 0]), 10, 10)

[2]: <ansys.geometry.core.sketch.sketch at 0x1d5223a5100>
```

#### Initiate design on server

436

Establish a server connection and initiate a design on the server.

```
[3]: modeler = launch_modeler()
design = modeler.create_design("ServiceColors")
```

#### Extrude the box sketch to create the matrix style design

Given the initial sketch, you can extrude it to create a matrix style design. In this example, you can create a 2x3 matrix of bodies. Each body is separated by 30 units in the X direction and 30 units in the Y direction. You have a total of 6 bodies.

#### Assign colors to the bodies

Given the previous design, you can assign a color to each body. You could have done this assignment while creating the bodies, but for the sake of encapsulating the color assignment logic, it is done in its own code cell.

```
[5]: colors = [["red", "blue", "yellow"], ["orange", "green", "purple"]]

for c_idx, comp in enumerate(design.components):
    for b_idx, body in enumerate(comp.bodies):
        body.color = colors[c_idx][b_idx]
        print(f"Body {body.name} has color {body.color}")

Body Component0_Body0 has color #ff0000
Body Component0_Body1 has color #0000ff
Body Component0_Body2 has color #ffff00
Body Component1_Body0 has color #ffa500
Body Component1_Body1 has color #008000
Body Component1_Body1 has color #008000
Body Component1_Body2 has color #800080
```

#### Plotting the design with colors

By default, the plot method does **not** use the colors assigned to the bodies. To plot the design with the assigned colors, you need to specifically request it.

Users have two options for plotting with the assigned colors:

- Pass the parameter use\_service\_colors=True to the plot method.
- Set the global parameter USE\_SERVICE\_COLORS to True.

It is important to note that the usage of colors when plotting might slow down the plotting process, as it requires additional information to be sent from the server to the client and processed in the client side.

If you just request the plot without setting the global parameter, the plot will be displayed without the colors, as shown below.

#### [6]: design.plot()

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
_http-equiv=&quot;Content-...
```

As stated previously, if you pass the parameter use\_service\_colors=True to the plot method, the plot is displayed with the assigned colors.

```
[7]: design.plot(use_service_colors=True)
```

```
\label{localize} Embeddable Widget(value='<iframe srcdoc=''<!DOCTYPE html>\n<html>\n <head>\n <meta\_ \\ \\ \\ \rightarrow http-equiv=\&quot;Content-...
```

However, if you set the global parameter to True, the plot is displayed with the assigned colors without the need to pass the parameter to the plot method.

```
[8]: import ansys.geometry.core as pyansys_geometry
```

```
pyansys_geometry.USE_SERVICE_COLORS = True
```

design.plot()

```
# Reverting the global parameter to its default value
```

```
pyansys_geometry.USE_SERVICE_COLORS = False
```

 $\label{localize} Embeddable \ Widget(value='<iframe srcdoc=''<!DOCTYPE html>\n<html>\n <head>\n <meta\_ http-equiv=&quot;Content-...$ 

This last method is useful when the user wants to plot all the designs with the assigned colors without the need to pass the parameter to the plot method in every call.

#### Plotting specific bodies or components with colors

If the user wants to plot specific bodies with the assigned colors, the user can follow the same approach as before. The user can pass the parameter use\_service\_colors=True to the plot method or set the global parameter USE\_SERVICE\_COLORS to True.

In the following examples, you are shown how to do this using the use\_service\_colors=True parameter.

Let's plot the first body of the first component with the assigned colors.

```
[9]: body = design.components[0].bodies[0]
```

```
body.plot(use_service_colors=True)
```

```
\label{localize} Embeddable \@widget(value='<iframe srcdoc=''<!DOCTYPE html>\n < head>\n < meta\_ \\ \rightarrow http-equiv=\&quot;Content-...
```

Now, let's plot the second component with the assigned colors.

#### [10]: comp = design.components[1]

```
comp.plot(use_service_colors=True)
```

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.13 Modeling: Surface bodies and trimmed surfaces

This example shows how to trim different surfaces, and how to use those surfaces to create surface bodies.

#### Create a surface

Create a sphere surface. This can be done without launching the modeler.

```
[1]: from ansys.geometry.core.shapes.surfaces import Sphere
from ansys.geometry.core.math import Point3D

surface = Sphere(origin=Point3D([0, 0, 0]), radius=1)
```

Now get information on how the surface is defined and parameterized.

- [2]: surface.parameterization()
- [2]: (Parameterization(form=ParamForm.PERIODIC, type=ParamType.CIRCULAR, interval=Interval(start=0, end=6.283185307179586)),
  Parameterization(form=ParamForm.CLOSED, type=ParamType.OTHER, interval=Interval(start=-1.5707963267948966, end=1.5707963267948966)))

#### Trim the surface

For a sphere, its parametization is (u: [0, 2\*pi], v:[-pi/2, pi/2]), where u corresponds to longitude and v corresponds to latitude. You can **trim** a surface by providing new parameters.

```
[3]: from ansys.geometry.core.shapes.box_uv import BoxUV
    from ansys.geometry.core.shapes.parameterization import Interval
    import math

trimmed_surface = surface.trim(BoxUV(range_u=Interval(0, math.pi), range_v=Interval(0, output))
```

From a TrimmedSurface, you can always refer back to the underlying Surface if needed.

- [4]: trimmed\_surface.geometry
- [4]: <ansys.geometry.core.shapes.surfaces.sphere.Sphere at 0x252a117f8f0>

#### Create a surface body

Now create a surface body by launching the modeler session and providing the trimmed surface. Then plot the body to see how you created a quarter of a sphere as a surface body.

```
[5]: from ansys.geometry.core import launch_modeler

modeler = launch_modeler()
print(modeler)

Ansys Geometry Modeler (0x252a91dfef0)

Ansys Geometry Modeler Client (0x252a9286d80)
    Target: localhost:700
    Connection: Healthy
```

If the sphere was left untrimmed, it would create a solid body since the surface is fully closed. In this case, since the surface was open, it created a surface body.

This same process can be used with other surfaces including:

- Cone
- Cylinder
- Plane
- Torus

Each surface has its own unique parameterization, which must be understood before trying to trim it.

#### **Close session**

When you finish interacting with your modeling service, you should close the active server session. This frees resources wherever the service is running.

Close the server session.

[7]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.14 Modeling: Using design parameters

You can read and update parameters that are part of the design. The simple design in this example has two associated parameters.

#### **Perform required imports**

```
[1]: from pathlib import Path
  import requests

from ansys.geometry.core import launch_modeler
```

The file for this example is in the integration tests folder and can be downloaded.

### Download the example file

Download the file for this example from the integration tests folder in the PyAnsys Geometry repository.

```
[2]: def download_file(url, filename):
        """Download a file from a URL and save it to a local file."""
        response = requests.get(url)
        response.raise_for_status() # Check if the request was successful
        with open(filename, 'wb') as file:
            file.write(response.content)
    # URL of the file to download
    url = "https://raw.githubusercontent.com/ansys/pyansys-geometry/main/tests/integration/
     →files/blockswithparameters.dsco"
    # Local path to save the file to
    file_path = Path.cwd() / "blockswithparameters.dsco"
    # Download the file
    download_file(url, file_path)
    print(f"File is downloaded to {file_path}")
    File is downloaded to C:\Users\ansys\actions-runner\_work\pyansys-geometry\pyansys-
     →geometry\doc\source\examples\03_modeling\blockswithparameters.dsco
```

#### Import a design with parameters

Import the model using the open\_file() method of the modeler.

#### Read existing parameters of the design

You can get all the parameters of the design as a list of parameters. Because this example has two parameters, you see two items in the list.

```
[4]: my_parameters = design.parameters
print(len(my_parameters))
2
```

A parameter object has a name, value, and unit.

```
[5]: print(my_parameters[0].name)
    print(my_parameters[0].dimension_value)
    print(my_parameters[0].dimension_type)

    print(my_parameters[1].name)
    print(my_parameters[1].dimension_value)
    print(my_parameters[1].dimension_type)

    p1
     0.0001087299999999982
    ParameterType.DIMENSIONTYPE_AREA
    p2
     0.0002552758322160814
    ParameterType.DIMENSIONTYPE_AREA
```

Parameter values are returned in the default unit for each dimension type. Since default length unit is meter and default area unit is meter square, the value is returned in square meters.

#### Edit a parameter value

You can edit the parameter's name or value by simply setting these fields. Set the second parameter (p2 value to 350 mm).

```
[6]: parameter1 = my_parameters[1]
  parameter1.dimension_value = 0.000440
  response = design.set_parameter(parameter1)
  print(response)
  print(my_parameters[0].dimension_value)
  print(my_parameters[1].dimension_value)

ParameterUpdateStatus.SUCCESS
  0.000108729999999982
  0.00044
```

After a successful parameter update, the design is updated. If we request the design plot again, we see the updated design.

The set\_parameter() method returns a Success status message if the parameter is updated or a "FAILURE" status message if the update fails. If the p2 parameter depends on the p1 parameter, updating the p1 parameter might also change the p2 parameter. In such cases, the method returns CONSTRAINED\_PARAMETERS, which indicates other parameters were also updated.

```
[8]: parameter1 = my_parameters[0]
  parameter1.dimension_value = 0.000250
  response = design.set_parameter(parameter1)
  print(response)
ParameterUpdateStatus.CONSTRAINED_PARAMETERS
```

To get the updated list, query the parameters once again.

```
[9]: my_parameters = design.parameters
    print(my_parameters[0].dimension_value)
    print(my_parameters[1].dimension_value)

    0.0002499999999997274
    0.0005812699999999445
```

#### Close the modeler

Close the modeler to free up resources and release the connection.

[10]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.3.15 Modeling: Chamfer edges and faces

A chamfer is an angled cut on an edge. Chamfers can be created using the Modeler.geometry\_commands module.

#### Create a block

Launch the modeler and create a block.

```
[1]: from ansys.geometry.core import launch_modeler, Modeler

modeler = Modeler()
print(modeler)

Ansys Geometry Modeler (0x228ffa0eb40)

Ansys Geometry Modeler Client (0x228ffc3d2e0)
    Target: 127.0.0.1:700
    Connection: Healthy
```

### **Chamfer edges**

Create a uniform chamfer on all edges of the block.

#### **Chamfer faces**

The chamfer of a face can also be modified. Create a chamfer on a single edge and then modify the chamfer distance value by providing the newly created face that represents the chamfer.

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.4 Applied examples

These examples demonstrate the usage of PyAnsys Geometry for real-world applications.

# 1 Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.4.1 Applied: Create a NACA 4-digit airfoil

NACA airfoils are a series of airfoil shapes for aircraft wings developed by the National Advisory Committee for Aeronautics (NACA). They are a standardized system of airfoil shapes that are defined by a series of digits. The digits, which indicate the shape of the airfoil, are used to create the airfoil shape.

Each digit in the NACA airfoil number has a specific meaning:

- The first digit defines the maximum camber as a percentage of the chord length.
- The second digit defines the position of the maximum camber as a percentage of the chord length.
- · The last two digits define the maximum thickness of the airfoil as a percentage of the chord length.

To fully understand the previous definitions, it is important to know that the chord length is the length of the airfoil from the leading edge to the trailing edge. The camber is the curvature of the airfoil, and the thickness is the distance between the upper and lower surfaces.

Symmetric airfoils have a camber of 0% and consequently, the first two digits of the NACA number are 0. For example, the NACA 0012 airfoil is a symmetric airfoil with a maximum thickness of 12%.

#### Define the NACA 4-digit airfoil equation

The following code uses the equation for a NACA 4-digit airfoil to create a set of points that define the airfoil shape. These points are Point2D objects in PyAnsys Geometry.

```
import numpy as np
from ansys.geometry.core.math import Point2D

def naca_airfoil_4digits(number: Union[int, str], n_points: int = 200) -> list[Point2D]:
    """
    Generate a NACA 4-digits airfoil.

Parameters
------
number : int or str
    NACA 4-digit number.
n_points : int
    Number of points to generate the airfoil. The default is ``200``.
    Number of points in the upper side of the airfoil.
    The total number of points is ``2 * n_points - 1``.

Returns
------
list[Point2D]
    List of points that define the airfoil.
"""
```

```
# Check if the number is a string
if isinstance(number, str):
   number = int(number)
# Calculate the NACA parameters
m = number // 1000 * 0.01
p = number // 100 \% 10 * 0.1
t = number \% 100 * 0.01
# Generate the airfoil
points = []
for i in range(n_points):
    # Make it a exponential distribution so the points are more concentrated
    # near the leading edge
   x = (1 - np.cos(i / (n_points - 1) * np.pi)) / 2
    # Check if it is a symmetric airfoil
    if p == 0 and m == 0:
        # Camber line is zero in this case
        yc = 0
        dyc_dx = 0
    else:
        # Compute the camber line
        if x < p:
           yc = m / p**2 * (2 * p * x - x**2)
            dyc_dx = 2 * m / p**2 * (p - x)
            yc = m / (1 - p) ** 2 * ((1 - 2 * p) + 2 * p * x - x**2)
            dyc_dx = 2 * m / (1 - p) ** 2 * (p - x)
    # Compute the thickness
    yt = 5 * t * (0.2969 * x**0.5)
                  - 0.1260 * x
                  - 0.3516 * x**2
                  + 0.2843 * x**3
                  -0.1015 * x**4)
    # Compute the angle
    theta = np.arctan(dyc_dx)
    # Compute the points (upper and lower side of the airfoil)
    xu = x - yt * np.sin(theta)
    yu = yc + yt * np.cos(theta)
    xl = x + yt * np.sin(theta)
   yl = yc - yt * np.cos(theta)
    # Append the points
   points.append(Point2D([xu, yu]))
   points.insert(0, Point2D([xl, yl]))
    # Remove the first point since it is repeated
```

```
if i == 0:
    points.pop(0)

return points
```

#### Example of a symmetric airfoil: NACA 0012

Now that the function for generating a NACA 4-digit airfoil is generated, this code creates a NACA 0012 airfoil, which is symmetric. This airfoil has a maximum thickness of 12%. The NACA number is a constant.

```
[2]: NACA_AIRFOIL = "0012"
```

#### **Required imports**

Before you start creating the airfoil points, you must import the necessary modules to create the airfoil using PyAnsys Geometry.

```
[3]: from ansys.geometry.core import launch_modeler from ansys.geometry.core.sketch import Sketch
```

#### Generate the airfoil points

Using the function defined previously, you generate the points that define the NACA 0012 airfoil. Create a Sketch object and add the points to it. Then, approximate the airfoil using straight lines between the points.

```
[4]: # Create a sketch
sketch = Sketch()

# Generate the points of the airfoil
points = naca_airfoil_4digits(NACA_AIRFOIL)

# Create the segments of the airfoil
for i in range(len(points) - 1):
    sketch.segment(points[i], points[i + 1])

# Close the airfoil
sketch.segment(points[-1], points[0])

# Plot the airfoil
sketch.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

#### Create the 3D airfoil

Once the Sketch object is created, you create a 3D airfoil. For this operation, you must create a modeler object, create a design object, and extrude the sketch.

```
[5]: # Launch the modeler
modeler = launch_modeler()
(continues on next page)
```

```
# Create the design
design = modeler.create_design(f"NACA_Airfoil_{NACA_AIRFOIL}")

# Extrude the airfoil
design.extrude_sketch("Airfoil", sketch, 1)

# Plot the design
design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

#### Save the design

Finally, save the design to a file. This file can be used in other applications or imported into a simulation software. This code saves the design as an FMD file, which can then be imported into Ansys Fluent.

#### Close the modeler

[7]: modeler.close()

#### Example of a cambered airfoil: NACA 6412

This code creates a NACA 6412 airfoil, which is cambered. This airfoil has a maximum camber of 6% and a maximum thickness of 12%. After overriding the NACA number, the code generates the airfoil points.

```
[8]: NACA_AIRFOIL = "6412"
```

#### Generate the airfoil points

As before, you generate the points that define the NACA 6412 airfoil. Create a Sketch object and add the points to it. Then, approximate the airfoil using straight lines.

```
[9]: # Create a sketch
sketch = Sketch()

# Generate the points of the airfoil
points = naca_airfoil_4digits(NACA_AIRFOIL)

# Create the segments of the airfoil
for i in range(len(points) - 1):
    sketch.segment(points[i], points[i + 1])

# Close the airfoil
```

```
sketch.segment(points[-1], points[0])

# Plot the airfoil
sketch.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

#### Create the 3D airfoil

```
[10]: # Launch the modeler
modeler = launch_modeler()

# Create the design
design = modeler.create_design(f"NACA_Airfoil_{NACA_AIRFOIL}")

# Extrude the airfoil
design.extrude_sketch("Airfoil", sketch, 1)

# Plot the design
design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

#### Save the design

In this case, the design is saved as an SCDOCX file.

#### Close the modeler

[12]: modeler.close()

#### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

### **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

# 8.4.2 Applied: Prepare a NACA airfoil for a Fluent simulation

Once a NACA airfoil is designed, it is necessary to prepare the geometry for a CFD simulation. This notebook demonstrates how to prepare a NACA 6412 airfoil for a Fluent simulation. The starting point of this example is the previously designed NACA 6412 airfoil. The airfoil was saved in an SCDOCX file, which is now imported into the notebook. The geometry is then prepared for the simulation.

In case you want to run this notebook, make sure that you have run the previous notebook to design the NACA 6412 airfoil.

#### Import the NACA 6412 airfoil

The following code starts up the Geometry Service and imports the NACA 6412 airfoil. The airfoil is then displayed in the notebook.

```
from ansys.geometry.core import launch_modeler

# Launch the modeler
modeler = launch_modeler()

# Import the NACA 6412 airfoil
design = modeler.open_file(os.path.join(os.getcwd(), f"NACA_Airfoil_6412.scdocx"))

# Retrieve the airfoil body
airfoil = design.bodies[0]

# Display the airfoil
design.plot()

EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html>\n <head>\n <meta_
http-equiv=&quot;Content-...</pre>
```

#### Prepare the geometry for the simulation

The current design is only composed of the airfoil. To prepare the geometry for the simulation, you must define the domain around the airfoil. The following code creates a rectangular fluid domain around the airfoil.

The airfoil has the following dimensions:

- Chord length: 1 (X-axis)
- Thickness: Depends on NACA value (Y-axis)

Define the fluid domain as a large box with these dimensions:

- Length (X-axis) 10 times the chord length
- Width (Z-axis) 5 times the chord length
- Height (Y-axis) 4 times the chord length

Place the airfoil at the center of the fluid domain.

```
[2]: from ansys.geometry.core.math import Point2D, Plane, Point3D
from ansys.geometry.core.sketch import Sketch

BOX_LENGTH = 10 # X-Axis
```

```
BOX_WIDTH = 5  # Z-Axis
BOX_HEIGHT = 4  # Y-Axis

# Create the sketch
fluid_sketch = Sketch(
    plane=Plane(origin=Point3D([0, 0, 0.5 - (BOX_WIDTH / 2)]))
)
fluid_sketch.box(
    center=Point2D([0.5, 0]),
    height=BOX_HEIGHT,
    width=BOX_LENGTH,
)

# Extrude the fluid domain
fluid = design.extrude_sketch("Fluid", fluid_sketch, BOX_WIDTH)
```

#### Create named selections

Named selections are used to define boundary conditions in Fluent. The following code creates named selections for the inlet, outlet, and walls of the fluid domain. The airfoil is also assigned a named selection.

The airfoil is aligned with the X axis. The inlet is located at the left side of the airfoil, the outlet is located at the right side of the airfoil, and the walls are located at the top and bottom of the airfoil. The inlet face has therefore a negative X-axis normal vector, and the outlet face has a positive X-axis normal vector. The rest of the faces, therefore, constitute the walls.

```
[3]: # Create named selections in the fluid domain (inlet, outlet, and surrounding faces)
    # Add also the airfoil as a named selection
    fluid_faces = fluid.faces
    surrounding_faces = []
    inlet_faces = []
    outlet_faces = []
    for face in fluid_faces:
        if face.normal().x == 1:
            outlet_faces.append(face)
        elif face.normal().x == -1:
            inlet_faces.append(face)
        else:
            surrounding_faces.append(face)
    design.create_named_selection("Outlet Fluid", faces=outlet_faces)
    design.create_named_selection("Inlet Fluid", faces=inlet_faces)
    design.create_named_selection("Surrounding Faces", faces=surrounding_faces)
    design.create_named_selection("Airfoil Faces", faces=airfoil.faces)
[3]: <ansys.geometry.core.designer.selection.NamedSelection at 0x1f669fdbf80>
```

#### Display the geometry

The geometry is now ready for the simulation. The following code displays the geometry in the notebook. This example uses the GeometryPlotter class to display the geometry for the airfoil and fluid domain in different colors with a specified opacity level.

The airfoil is displayed in green, and the fluid domain is displayed in blue with an opacity of 0.25.

#### **Export the geometry**

Export the geometry into a Fluent-compatible format. The following code exports the geometry into a PMDB file, which retains the named selections.

```
[5]: # Save the design
file = design.export_to_pmdb()
print(f"Design saved to {file}.")

Design saved to C:\Users\ansys\actions-runner\_work\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geometry\pyansys-geome
```

You can import the exported PMDB file into Fluent to set up the mesh and perform the simulation. For an example of how to set up the mesh and boundary conditions in Fluent, see the Modeling External Compressible Flow example in the Fluent documentation.

The main difference between the Fluent example and this geometry is the coordinate system. The Fluent example defines the airfoil in the XY plane, while this geometry defines the airfoil in the XZ plane.

#### Close the modeler

[6]: modeler.close()

# **1** Download this example

Download this example as a Jupyter Notebook or as a Python script.

**CHAPTER** 

NINE

## **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 PyAnsys Geometry.

The following contribution information is specific to PyAnsys Geometry.

# 9.1 Clone the repository

To clone and install the latest PyAnsys Geometry release in development mode, run these commands:

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

### 9.2 Post issues

Use the PyAnsys Geometry Issues page to submit questions, report bugs, and request new features. When possible, you should use these issue templates:

- Bug, problem, error: For filing a bug report
- Documentation error: For requesting modifications to the documentation
- Adding an example: For proposing a new example
- New feature: For requesting enhancements to the code

If your issue does not fit into one of these template categories, you can click the link for opening a blank issue.

To reach the project support team, email pyansys.core@ansys.com.

### 9.3 View documentation

Documentation for the latest stable release of PyAnsys Geometry is hosted at PyAnsys Geometry Documentation.

In the upper right corner of the documentation's title bar, there is an option for switching from viewing the documentation for the latest stable release to viewing the documentation for the development version or previously released versions.

# 9.4 Adhere to code style

PyAnsys Geometry 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
```

This way, it's not possible for you to push code that fails the style checks:

# 9.5 Build the documentation

#### Note

To build the documentation, you must have the Geometry Service installed and running on your machine because it is used to generate the examples in the documentation. It is also recommended that the service is running as a Docker container.

If you do not have the Geometry Service installed, you can still build the documentation, but the examples are not generated. To build the documentation without the examples, define the following environment variable:

```
# On Linux or macOS
export BUILD_EXAMPLES=false

# On Windows CMD
set BUILD_EXAMPLES=false

# On Windows PowerShell
$env:BUILD_EXAMPLES="false"
```

To build the documentation locally, you must run this command to install the documentation dependencies:

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

Then, navigate to the docs directory and run this command:

```
# On Linux or macOS
make html

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

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

You can clean the documentation build by running this command:

```
# On Linux or macOS
make clean
# On Windows
./make.bat clean
```

# 9.6 Adding examples

Users can collaborate with examples to this documentation by adding new examples. A reference commit of the changes that adding an example requires is shown here:

https://github.com/ansys/pyansys-geometry/pull/1454/commits/7fcf02f86f05e0e5ce1c1071c3c5fcd274ec481c

To add a new example, follow these steps:

- 1. Create a new notebook in the doc/source/examples directory, under the appropriate folder for your example.
- 2. Take as reference an existing example to see how the notebook should be structured.
- 3. Add the new notebook to the doc/source/examples.rst file.
- 4. Store a thumbnail image of the example in the doc/source/\_static/thumbnails directory.
- 5. Link the thumbnail image to your example file in doc/source/conf.py as shown in the reference commit.

You can also test the correct build process of a new example by performing the following steps:

1. Run the following command to install the documentation dependencies:

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

2. Install myst-nb by running this command:

```
pip install myst-nb
```

3. Run the following command to build a single example (that is, substitute <PATH\_TO\_MY\_EXAMPLE\_FILE> with the path to your example file):

```
mystnb-docutils-html --nb-read-as-md=1 <PATH_TO_MY_EXAMPLE_FILE> output.html
```

4. Check the output file output.html to ensure that the example is correctly built. Rendered output does not have documentation styling but users should have the ability to see its proper execution.

### 1 Note

Plots are not be rendered in the output file, but the code and markdown cells should be correctly rendered. In case of failure during execution users can also see the error message in the output file.

# 9.7 Run tests

PyAnsys Geometry uses pytest for testing.

# 9.7.1 Prerequisites

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

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

Make sure to define the port and host of the service using the following environment variables:

```
# On Linux or macOS
export ANSRV_GEO_PORT=5000
export ANSRV_GEO_HOST=localhost

# On Windows CMD
set ANSRV_GEO_PORT=5000
set ANSRV_GEO_HOST=localhost

# On Windows PowerShell
$env:ANSRV_GEO_PORT=5000
$env:ANSRV_GEO_HOST="localhost"
```

# 9.7.2 Running the tests

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

pytest



The tests require the Geometry Service to be installed and running on your machine. The tests fail if the service is not running. It is expected for the Geometry Service to be running as a Docker container.

If you do not have the Geometry Service running as a Docker container, but you have it running on your machine, you can still run the tests with the following argument:

```
pytest --use-existing-service=yes
```

**CHAPTER** 

TEN

## **ASSETS**

In this section, users are able to download a set of assets related to PyAnsys Geometry.

# 10.1 Documentation

The following links provide users with downloadable documentation in various formats

- Documentation in HTML format
- Documentation in PDF format

# 10.2 Wheelhouse

If you lack an internet connection on your installation machine, you should install PyAnsys Geometry by downloading the wheelhouse archive.

Each wheelhouse archive contains all the Python wheels necessary to install PyAnsys Geometry from scratch on Windows, Linux, and MacOS from Python 3.10 to 3.13. You can install this on an isolated system with a fresh Python installation or on a virtual environment.

For example, on Linux with Python 3.10, unzip the wheelhouse archive and install it with:

```
unzip ansys-geometry-core-v0.9.1-all-wheelhouse-ubuntu-latest-3.10.zip wheelhouse pip install ansys-geometry-core -f wheelhouse --no-index --upgrade --ignore-installed
```

If you are on Windows with Python 3.10, unzip to a wheelhouse directory by running -d wheelhouse (this is required for unzipping to a directory on Windows) and install using the preceding command.

Consider installing using a virtual environment.

The following wheelhouse files are available for download:

#### 10.2.1 Linux

- Linux wheelhouse for Python 3.10
- Linux wheelhouse for Python 3.11
- Linux wheelhouse for Python 3.12
- Linux wheelhouse for Python 3.13

### 10.2.2 Windows

- Windows wheelhouse for Python 3.10
- Windows wheelhouse for Python 3.11
- Windows wheelhouse for Python 3.12
- Windows wheelhouse for Python 3.13

#### 10.2.3 MacOS

- MacOS wheelhouse for Python 3.10
- MacOS wheelhouse for Python 3.11
- MacOS wheelhouse for Python 3.12
- MacOS wheelhouse for Python 3.13

# 10.3 Geometry service Docker container assets

Build the latest Geometry service Docker container using the following assets. For information on how to build the container, see *Docker containers*.

Currently, the Geometry service backend is mainly delivered as a **Windows** Docker container. However, these containers require a Windows machine to run them.

#### 10.3.1 Windows container



Only users with access to https://github.com/ansys/pyansys-geometry-binaries can download these binaries.

- · Latest Geometry service binaries for Windows containers
- Latest Geometry service Dockerfile for Windows containers

458 Chapter 10. Assets

# **ELEVEN**

# **RELEASE NOTES**

This document contains the release notes for the PyAnsys Geometry project.

### 11.1 0.9.1 - 2025-02-28

#### 11.1.1 Added

- offset faces set radius implementation + testing #1769
- separate graphics target #1782

# 11.1.2 Dependencies

- bump the docs-deps group with 2 updates #1762
- bump ansys-api-geometry from 0.4.38 to 0.4.40 #1763
- bump ansys-sphinx-theme[autoapi] from 1.3.1 to 1.3.2 in the docs-deps group #1766
- bump ansys-tools-visualization-interface from 0.8.1 to 0.8.3 #1767
- bump sphinx from 8.2.0 to 8.2.1 in the docs-deps group #1772
- bump ansys-api-geometry from 0.4.40 to 0.4.42 #1773
- temporary workaround for using trusted publisher approach #1783

#### 11.1.3 Fixed

- allow setting max message length for send operations #1775
- typo in labeler.yml file #1776
- docker build process failing on helper script #1785

#### 11.1.4 Maintenance

- bump dev version to 0.10.dev0 #1752
- update CHANGELOG for v0.9.0 #1760
- pre-commit automatic update #1770

# 11.2 0.9.0 - 2025-02-18

#### 11.2.1 Added

- design activation changes #1707
- add contributors #1708
- Implementation of inspect & repair geometry #1712
- launch core service from envar #1716
- workflow enhancements for better tool results #1723
- add face color, round info, bring measure tools to linux #1732
- conservative approach to single design per modeler #1740
- export glb #1741
- allow plotting of individual faces #1757

# 11.2.2 Dependencies

- bump ansys-api-geometry from 0.4.33 to 0.4.34 #1709
- bump ansys-sphinx-theme[autoapi] from 1.2.6 to 1.2.7 in the docs-deps group #1719
- bump ansys-api-geometry from 0.4.34 to 0.4.35 #1720
- bump ansys-sphinx-theme[autoapi] from 1.2.7 to 1.3.0 in the docs-deps group #1726
- bump ansys-sphinx-theme[autoapi] from 1.3.0 to 1.3.1 in the docs-deps group #1728
- bump ansys-api-geometry from 0.4.35 to 0.4.36 #1729
- bump trame-vtk from 2.8.14 to 2.8.15 #1736
- bump jupytext from 1.16.6 to 1.16.7 in the docs-deps group #1742
- bump ansys-api-geometry from 0.4.36 to 0.4.37 #1743
- bump myst-parser from 4.0.0 to 4.0.1 in the docs-deps group #1744
- bump ansys-api-geometry from 0.4.37 to 0.4.38 #1746
- bump numpy from 2.2.2 to 2.2.3 #1747
- bump panel from 1.6.0 to 1.6.1 #1749
- bump scipy from 1.15.1 to 1.15.2 #1756

### 11.2.3 Documentation

• update CONTRIBUTING.md #1730

#### 11.2.4 Fixed

- re enable fmd tests #1711
- support body mirror on linux #1714
- use sketch plane for imprint/project curves #1715
- revert boolean ops logic and hold-off on commands-based implementation (temporarily) #1725
- Linux Core Service docker file was missing a dependency #1758

### 11.2.5 Maintenance

- update CHANGELOG for v0.8.2 #1706
- pre-commit automatic update #1717, #1737
- update SECURITY.md versions supported #1722
- keep simba-plugin-geometry tag #1739
- enhancements to GLB export and object plot() methods #1750
- clean up deprecation warning for trapezoid class and add more info on deprecation #1754

#### 11.2.6 Test

- verifying issue with empty intersect and temporal body creation #1258
- Expand pattern tests #1713
- set body name #1727
- activate 8 linux tests #1745

# 11.3 0.8.2 - 2025-01-29

### 11.3.1 Added

- create a fillet on an edge/face #1621
- create a full fillet between multiple faces #1623
- extrude existing faces, setup face offset relationships #1628
- interference repair tool #1633
- extrude existing edges to create surface bodies #1638
- create and modify linear patterns #1641
- body suppression state #1643
- parameters refurbished #1647
- rename object #1648
- surface body from trimmed curves #1650
- create circular and fill patterns #1653
- find fix simplify #1661
- replace face #1664
- commands for merge and intersect #1665
- revolve faces a set distance, up to another object, or by a helix #1666
- add split body and tests #1669
- enable get/set persistent ids for stride import/export #1671
- find and fix edge methods #1672
- shell methods #1673
- implementation of NURBS curves #1675

- get assigned material #1684
- matrix rotation and translation #1689
- is\_core\_service BackendType static method #1692
- export and download stride format #1698
- blitz development #1701

# 11.3.2 Dependencies

- bump ansys-tools-visualization-interface from 0.7.0 to 0.8.1 #1640
- bump ansys-api-geometry from 0.4.27 to 0.4.28 #1644
- bump sphinx-autodoc-typehints from 3.0.0 to 3.0.1 in the docs-deps group #1651
- bump ansys-api-geometry from 0.4.28 to 0.4.30 #1652
- bump protobuf from 5.28.3 to 5.29.3 in the grpc-deps group across 1 directory #1656
- bump numpy from 2.2.1 to 2.2.2 #1662
- bump ansys-api-geometry from 0.4.30 to 0.4.31 #1663
- bump ansys api geometry from 0.4.30 to 0.4.32 #1677
- bump ansys-api-geometry from 0.4.31 to 0.4.32 #1681
- bump panel from 1.5.5 to 1.6.0 #1682
- bump semver from 3.0.2 to 3.0.4 #1687
- bump ansys-api-geometry from 0.4.32 to 0.4.33 #1695
- bump nbconvert from 7.16.5 to 7.16.6 in the docs-deps group #1700

#### 11.3.3 Fixed

- reactivate test on failing extra edges test #1396
- filter set export id to only CoreService based backends #1685
- cleanup unsupported module #1690
- disable unimplemented tests #1691
- tech review fixes for blitz branch #1703

#### 11.3.4 Maintenance

- update CHANGELOG for v0.8.1 #1639
- whitelist semver package temporarily #1657
- reverting semver package whitelist since problematic version is yanked #1659
- pre-commit automatic update #1667, #1696
- ensure design is closed on test exit #1680
- use dedicate pygeometry-ci-2 runner #1693
- remove towncrier info duplicates #1702

### 11.3.5 Test

- add more find and fix tests for repair tools #1645
- Add some new tests #1670
- add unit tests for 3 repair tools #1683

### 11.4 0.8.1 - 2025-01-15

# 11.4.1 Dependencies

• bump ansys-api-geometry from 0.4.26 to 0.4.27 #1634

#### 11.4.2 Fixed

• release issues encountered #1637

#### 11.4.3 Maintenance

• update CHANGELOG for v0.8.0 #1636

## 11.5 0.8.0 - 2025-01-15

#### 11.5.1 Added

- active support for Python 3.13 #1481
- add chamfer tool #1495
- allow version input to automatically consider the nuances for the Ansys Student version #1548
- adapt health check timeout algorithm #1559
- add core service support #1571
- enable (partially) prepare and repair tools in Core Service #1580
- create launcher for core services #1587

# 11.5.2 Dependencies

- bump ansys-api-geometry from 0.4.16 to 0.4.17 #1547
- bump ansys-sphinx-theme[autoapi] from 1.2.1 to 1.2.2 in the docs-deps group #1549
- bump ansys-api-geometry from 0.4.17 to 0.4.18 #1550
- bump ansys-tools-visualization-interface from 0.5.0 to 0.6.0 #1554
- bump pytest from 8.3.3 to 8.3.4 #1562
- bump six from 1.16.0 to 1.17.0 #1568
- bump the docs-deps group across 1 directory with 2 updates #1570
- bump ansys-api-geometry from 0.4.18 to 0.4.20 #1574
- bump numpy from 2.1.3 to 2.2.0 #1575
- bump ansys-api-geometry from 0.4.20 to 0.4.23 #1581
- bump ansys-api-geometry from 0.4.23 to 0.4.24 #1582

- bump ansys-tools-visualization-interface from 0.6.0 to 0.6.1 #1583
- bump ansys-tools-visualization-interface from 0.6.1 to 0.6.2 #1586
- avoid the usage of attrs 24.3.0 (temporary) #1589
- bump jupytext from 1.16.4 to 1.16.5 in the docs-deps group #1590
- bump jupytext from 1.16.5 to 1.16.6 in the docs-deps group #1593
- bump panel from 1.5.4 to 1.5.5 #1595
- bump ansys-sphinx-theme[autoapi] from 1.2.3 to 1.2.4 in the docs-deps group #1597
- bump notebook from 7.3.1 to 7.3.2 in the docs-deps group #1598
- bump numpy from 2.2.0 to 2.2.1 #1599
- bump ansys-tools-path from 0.7.0 to 0.7.1 #1600
- bump nbsphinx from 0.9.5 to 0.9.6 in the docs-deps group #1602
- bump nbconvert from 7.16.4 to 7.16.5 in the docs-deps group #1609
- bump ansys-api-geometry from 0.4.24 to 0.4.25 #1610
- bump sphinx-autodoc-typehints from 2.5.0 to 3.0.0 in the docs-deps group #1611
- bump scipy from 1.14.1 to 1.15.0 #1612
- bump trame-vtk from 2.8.12 to 2.8.13 #1616
- bump trame-vtk from 2.8.13 to 2.8.14 #1617
- bump ansys-tools-visualization-interface from 0.6.2 to 0.7.0 #1619
- bump ansys-sphinx-theme[autoapi] from 1.2.4 to 1.2.6 in the docs-deps group #1624
- bump scipy from 1.15.0 to 1.15.1 #1625
- bump ansys-api-geometry from 0.4.25 to 0.4.26 #1626

#### 11.5.3 Documentation

• Explain how to report a security issue. #1605

# 11.5.4 Fixed

- numpydoc warnings #1556
- vtk/pyvista issues #1584
- make\_child\_logger only takes 2 args. #1603
- FAQ on install #1631

#### 11.5.5 Maintenance

- pre-commit automatic update #1366, #1552, #1561, #1588, #1601, #1615, #1630
- update CHANGELOG for v0.7.6 #1545
- change release artifacts self-hosted runner #1546
- automerge pre-commit.ci PRs #1553
- bump pyvista/setup-headless-display-action to v3 #1555

- decouple unstable image promotion #1591
- skip unnecessary stages when containers are the same #1592
- Numpy is already imported at the top of the module. #1604
- update license year using pre-commit hook #1608

# 11.6 0.7.6 - 2024-11-19

#### 11.6.1 Added

• allow for some additional extrusion direction names #1534

# 11.6.2 Dependencies

- bump ansys-sphinx-theme[autoapi] from 1.1.7 to 1.2.0 in the docs-deps group #1520
- bump ansys-tools-visualization-interface from 0.4.7 to 0.5.0 #1521
- bump numpy from 2.1.2 to 2.1.3 #1522
- bump ansys-api-geometry from 0.4.13 to 0.4.14 #1525
- bump ansys-api-geometry from 0.4.14 to 0.4.15 #1529
- bump pint from 0.24.3 to 0.24.4 #1530
- bump trame-vtk from 2.8.11 to 2.8.12 #1531
- bump ansys-sphinx-theme[autoapi] from 1.2.0 to 1.2.1 in the docs-deps group #1535
- bump panel from 1.5.3 to 1.5.4 #1536
- bump ansys-tools-path from 0.6.0 to 0.7.0 #1537
- bump ansys-api-geometry from 0.4.15 to 0.4.16 #1538
- limit upper version on grpcio & grpcio-health-checking to 1.68 #1544

#### 11.6.3 Documentation

- typo with the docstrings #1524
- change max header links before more dropdown #1527

#### 11.6.4 Maintenance

- update CHANGELOG for v0.7.5 #1519
- pre-commit automatic update #1523, #1532, #1543
- bump codecov/codecov-action from 4 to 5 in the actions group #1541

# 11.7 0.7.5 - 2024-10-31

#### 11.7.1 Added

- create body from surface #1454
- performance enhancements to plotter #1496
- allow picking from easy access methods #1499

- implement cut operation in extrude sketch #1510
- caching bodies to avoid unnecessary object creation #1513
- enable retrieval of service logs (via API) #1515

### 11.7.2 Dependencies

- bump sphinx from 8.1.0 to 8.1.3 in the docs-deps group #1479
- bump ansys-sphinx-theme[autoapi] from 1.1.4 to 1.1.5 in the docs-deps group #1482
- bump the grpc-deps group across 1 directory with 3 updates #1487
- bump ansys-sphinx-theme[autoapi] from 1.1.5 to 1.1.6 in the docs-deps group #1493
- bump trame-vtk from 2.8.10 to 2.8.11 #1494
- bump ansys-api-geometry from 0.4.11 to 0.4.12 #1502
- bump protobuf from 5.28.2 to 5.28.3 in the grpc-deps group #1505
- bump ansys-sphinx-theme[autoapi] from 1.1.6 to 1.1.7 in the docs-deps group #1506
- bump ansys-tools-visualization-interface from 0.4.6 to 0.4.7 #1507
- bump panel from 1.5.2 to 1.5.3 #1508
- bump ansys-api-geometry from 0.4.12 to 0.4.13 #1512
- bump the grpc-deps group with 2 updates #1517
- bump pytest-cov from 5.0.0 to 6.0.0 #1518

#### 11.7.3 Documentation

- avoid having a drop down in the top navigation bar #1485
- provide information on how to build a single example #1490
- add example file to download in the test #1501
- revisit examples to make sure they are properly styled #1509
- align landing page layout with UI/UX requirements #1511

#### 11.7.4 Fixed

- static search options #1478
- respect product\_version when launching geometry service #1486

#### 11.7.5 Maintenance

- update CHANGELOG for v0.7.4 #1476
- pre-commit automatic update #1480, #1516
- avoid linkcheck on changelog (unnecessary) #1489
- update CONTRIBUTORS #1492
- allowing new tags for Windows Core Service #1497
- simplify vulnerabilities check #1504

## 11.8 0.7.4 - 2024-10-11

## 11.8.1 Dependencies

- bump sphinx from 8.0.2 to 8.1.0 in the docs-deps group #1470
- bump ansys-api-geometry from 0.4.10 to 0.4.11 #1473
- bump ansys-sphinx-theme to v1.1.3 #1475

#### 11.8.2 Fixed

- solving intersphinx warnings on paths #1469
- check\_input\_types not working with forward refs #1471
- share\_topology is available on 24R2 #1472

#### 11.8.3 Maintenance

• update CHANGELOG for v0.7.3 #1466

# 11.9 0.7.3 - 2024-10-09

### 11.9.1 Added

- use service colors in plotter (upon request) #1376
- capability to close designs (also on modeler.exit()) #1409
- prioritize user-defined SPACECLAIM MODE env var #1440
- verifying Linux service also accepts colors #1451

### 11.9.2 Dependencies

- bump protobuf from 5.28.0 to 5.28.1 in the grpc-deps group #1424
- bump the docs-deps group with 2 updates #1425, #1436
- bump ansys-tools-visualization-interface from 0.4.3 to 0.4.4 #1426
- bump pytest from 8.3.2 to 8.3.3 #1427
- bump panel from 1.4.5 to 1.5.0 #1428
- bump protobuf from 5.28.1 to 5.28.2 in the grpc-deps group #1435
- bump the grpc-deps group with 3 updates #1442
- bump beartype from 0.18.5 to 0.19.0 #1443
- bump panel from 1.5.0 to 1.5.1 #1444
- bump ansys-sphinx-theme[autoapi] from 1.1.1 to 1.1.2 in the docs-deps group #1456
- bump ansys-api-geometry from 0.4.8 to 0.4.9 #1457
- bump numpy from 2.1.1 to 2.1.2 #1458
- bump panel from 1.5.1 to 1.5.2 #1459
- bump ansys-api-geometry from 0.4.9 to 0.4.10 #1461

11.8. 0.7.4 - 2024-10-11 467

- bump ansys-tools-visualization-interface from 0.4.4 to 0.4.5 #1462
- update protobuf from 5.27.2 to 5.27.5 #1464
- bump sphinx-autodoc-typehints from 2.4.4 to 2.5.0 in the docs-deps group #1465

#### 11.9.3 Documentation

- adding cheat sheet on documentation #1433
- add captions in examples toctrees #1434

#### 11.9.4 Fixed

- ci/cd issues on documentation build #1441
- adapt tessellate tests to new core service #1449
- rename folders on Linux docker image according to new version #1450

#### 11.9.5 Maintenance

- update CHANGELOG for v0.7.2 #1422
- checkout LFS files from previous version to ensure upload #1423
- pre-commit automatic update #1431, #1437, #1445, #1460
- update to ansys actions v8 and docs theme (static search) #1446
- pyvista/setup-headless-display started failing #1447
- check method implemented in Ansys actions #1448
- unstable image promotion and dependabot daily updates #1463

## 11.10 0.7.2 - 2024-09-11

#### 11.10.1 Added

- allow for platform input when using Ansys Lab #1416
- ensure GrpcClient class closure upon deletion #1417

### 11.10.2 Dependencies

- bump sphinx-autodoc-typehints from 2.3.0 to 2.4.0 in the docs-deps group #1411
- bump numpy from 2.1.0 to 2.1.1 #1412
- bump ansys-tools-visualization-interface from 0.4.1 to 0.4.3 #1413

### 11.10.3 Documentation

- remove title from landing page #1408
- adapt examples to use launch\_modeler instead of Modeler obj connection #1410

### 11.10.4 Fixed

- handle properly np.cross() 2d ops deprecated in Numpy 2.X #1419
- change logo link so that it renders properly on PyPI #1420
- wrong path on logo image #1421

### 11.10.5 Maintenance

- update CHANGELOG for v0.7.1 #1407
- pre-commit automatic update #1418

### 11.11 0.7.1 - 2024-09-06

### 11.11.1 Added

- get and set body color #1357
- add modeler.exit() method #1375
- setting instance name during component creation #1382
- accept pathlib.Path as input in missing methods #1385
- default logs folder on Geometry Service started by Python at PUBLIC (Windows) #1386
- allowing users to specify API version when running script against SpaceClaim or Discovery #1395
- expose modeler.designs attribute #1401
- pretty print components #1403

### 11.11.2 Dependencies

- bump the grpc-deps group with 2 updates #1363, #1369
- bump the docs-deps group with 2 updates #1364, #1392
- bump numpy from 2.0.1 to 2.1.0 #1365
- bump ansys-sphinx-theme[autoapi] from 1.0.5 to 1.0.7 in the docs-deps group #1370
- bump ansys-api-geometry from 0.4.7 to 0.4.8 #1371
- bump scipy from 1.14.0 to 1.14.1 #1372
- bump the grpc-deps group with 3 updates #1391
- bump ansys-tools-visualization-interface from 0.4.0 to 0.4.1 #1393
- bump ansys-sphinx-theme[autoapi] from 1.0.7 to 1.0.8 in the docs-deps group #1397

### 11.11.3 Documentation

add project logo #1405

#### 11.11.4 Fixed

remove server\_logs\_folder argument for Discovery and SpaceClaim #1387

### 11.11.5 Maintenance

- update CHANGELOG for v0.7.0 #1360
- bump dev branch to v0.8.dev0 #1361
- solving various warnings #1368
- pre-commit automatic update #1373, #1394
- upload coverage artifacts properly with upload-artifact@v4.4.0 #1406

## 11.12 0.7.0 - 2024-08-13

### 11.12.1 Added

- build: drop support for Python 3.9 #1341
- feat: adapting beartype typehints to +Python 3.10 standard #1347

## 11.12.2 Dependencies

- build: bump the grpc-deps group with 3 updates #1342
- build: bump panel from 1.4.4 to 1.4.5 #1344
- bump the docs-deps group across 1 directory with 4 updates #1353
- bump trame-vtk from 2.8.9 to 2.8.10 #1355
- bump ansys-api-geometry from 0.4.6 to 0.4.7 #1356

#### 11.12.3 Documentation

• feat: update conf for version 1.x of ansys-sphinx-theme #1351

### 11.12.4 Fixed

• trapezoid signature change and internal checks #1354

#### 11.12.5 Maintenance

- updating Ansys actions to v7 changelog related #1348
- ci: bump ansys/actions from 6 to 7 in the actions group #1352
- pre-commit automatic update #1358

### 11.12.6 Miscellaneous

• chore: pre-commit automatic update #1345

### 11.13 0.6.6 - 2024-08-01

### 11.13.1 Added

- feat: Add misc. repair and prepare tool methods #1293
- feat: name setter and fill style getter setters #1299
- feat: extract fluid volume from solid #1306

- feat: keep "other" bodies when performing bool operations #1311
- feat: revolve\_sketch rotation definition enhancement #1336

## 11.13.2 Changed

- chore: update CHANGELOG for v0.6.5 #1290
- chore: enable ruff formatter on pre-commit #1312
- chore: updating dependabot groups #1313
- chore: adding issue links to TODOs #1320
- feat: adapt to new ansys-tools-visualization-interface v0.4.0 #1338

#### 11.13.3 Fixed

- test: create sphere bug raised after box creation #1291
- ci: docker cleanup #1294
- fix: default length units not being used properly on arc creation #1310

## 11.13.4 Dependencies

- build: bump ansys-api-geometry from 0.4.4 to 0.4.5 #1292
- build: bump pyvista[jupyter] from 0.43.10 to 0.44.0 in the docs-deps group #1296
- build: bump jupytext from 1.16.2 to 1.16.3 in the docs-deps group #1300
- build: bump ansys-api-geometry from 0.4.5 to 0.4.6 #1301
- build: bump pint from 0.24.1 to 0.24.3 #1307
- build: bump grpcio-health-checking from 1.60.0 to 1.64.1 in the grpc-deps group #1315
- build: bump the docs-deps group across 1 directory with 2 updates #1316
- build: bump the grpc-deps group with 2 updates #1322
- build: bump the docs-deps group with 2 updates #1323
- build: bump pyvista[jupyter] from 0.44.0 to 0.44.1 #1324
- build: bump ansys-tools-visualization-interface from 0.2.6 to 0.3.0 #1325
- build: bump pytest from 8.2.2 to 8.3.1 #1326
- build: bump pytest from 8.3.1 to 8.3.2 #1331
- build: bump numpy from 2.0.0 to 2.0.1 #1332

#### 11.13.5 Miscellaneous

• chore: pre-commit automatic update #1327, #1333

### 11.14 0.6.5 - 2024-07-02

### 11.14.1 Changed

- chore: update CHANGELOG for v0.6.4 #1278
- build: update sphinx-autodoc-typehints version #1280

• chore: update SECURITY.md #1286

#### 11.14.2 Fixed

• fix: manifest path should render as posix rather than uri #1289

## 11.14.3 Dependencies

- build: bump protobuf from 5.27.1 to 5.27.2 in the grpc-deps group #1283
- build: bump scipy from 1.13.1 to 1.14.0 #1284
- build: bump vtk from 9.3.0 to 9.3.1 #1287

#### 11.14.4 Miscellaneous

• chore: pre-commit automatic update #1281, #1288

# 11.15 0.6.4 - 2024-06-24

#### 11.15.1 Added

• feat: using ruff as the main linter/formatter #1274

## 11.15.2 Changed

- chore: update CHANGELOG for v0.6.3 #1273
- chore: bump pre-commit-hook version #1276

#### 11.15.3 Fixed

• fix: backticks breaking doc build after ruff linter #1275

# 11.15.4 Dependencies

• build: bump pint from 0.24 to 0.24.1 #1277

# 11.16 0.6.3 - 2024-06-18

## 11.16.1 Changed

- chore: update CHANGELOG for v0.6.2 #1263
- build: adapting to numpy 2.x #1265
- docs: using ansys actions (again) to build docs #1270

### 11.16.2 Fixed

- fix: unnecessary Point3D comparison #1264
- docs: examples are not being uploaded as assets (.py/.ipynb) #1268
- fix: change action order #1269

## 11.16.3 Dependencies

- build: bump numpy from 1.26.4 to 2.0.0 #1266
- build: bump the docs-deps group with 2 updates #1271

#### 11.16.4 Miscellaneous

• chore: pre-commit automatic update #1267

### 11.17 0.6.2 - 2024-06-17

#### 11.17.1 Added

- feat: deprecating log\_level and logs\_folder + adding client log control #1260
- feat: adding deprecation support for args and methods #1261

## 11.17.2 Changed

- chore: update CHANGELOG for v0.6.1 #1256
- ci: simplify doc build using ansys/actions #1262

### 11.17.3 Fixed

• fix: Rename built in shadowing variables #1257

# 11.18 0.6.1 - 2024-06-12

#### 11.18.1 Added

• feat: revolve a sketch given an axis and an origin #1248

# 11.18.2 Changed

- chore: update CHANGELOG for v0.6.0 #1245
- chore: update dev version to 0.8.dev0 #1246

### 11.18.3 Fixed

• fix: Bug in show function #1255

### 11.18.4 Dependencies

- build: bump protobuf from 5.27.0 to 5.27.1 in the grpc-deps group #1250
- build: bump the docs-deps group with 2 updates #1251
- build: bump trame-vtk from 2.8.8 to 2.8.9 #1252
- build: bump pint from 0.23 to 0.24 #1253
- build: bump ansys-tools-visualization-interface from 0.2.2 to 0.2.3 #1254

### 11.18.5 Miscellaneous

• docs: add conda information for package #1247

### 11.19 0.6.0 - 2024-06-07

### 11.19.1 Added

- feat: Adapt to ansys-visualizer #959
- fix: rename GeomPlotter to GeometryPlotter #1227
- refactor: use ansys-tools-visualization-interface global vars rather than env vars #1230
- feat: bump to use v251 as default #1242

## 11.19.2 Changed

- chore: update CHANGELOG for v0.5.6 #1213
- chore: update SECURITY.md #1214
- ci: use Trusted Publisher for releasing package #1216
- ci: remove pygeometry-ci-1 specific logic #1221
- ci: only run doc build on runners outside the ansys network #1223
- chore: pre-commit automatic update #1224
- ci: announce nightly workflows failing #1237
- ci: failing notifications improvement #1243
- fix: broken interactive docs and improved tests paths #1244

### 11.19.3 Fixed

- fix: Interactive documentation #1226
- fix: only notify on failure and fill with data #1238

### 11.19.4 Dependencies

- build: bump protobuf from 5.26.1 to 5.27.0 in the grpc-deps group #1217
- build: bump panel from 1.4.2 to 1.4.3 in the docs-deps group #1218
- build: bump ansys-api-geometry from 0.4.1 to 0.4.2 #1219
- build: bump ansys-sphinx-theme[autoapi] from 0.16.2 to 0.16.5 in the docs-deps group #1231
- build: bump requests from 2.32.2 to 2.32.3 #1232
- build: bump ansys-api-geometry from 0.4.2 to 0.4.3 #1233
- build: bump ansys-tools-visualization-interface from 0.2.1 to 0.2.2 #1234
- build: bump panel from 1.4.3 to 1.4.4 in the docs-deps group #1235
- build: bump ansys-tools-path from 0.5.2 to 0.6.0 #1236
- build: bump grpcio from 1.64.0 to 1.64.1 in the grpc-deps group #1239
- build: bump ansys-api-geometry from 0.4.3 to 0.4.4 #1240

• build: bump pytest from 8.2.1 to 8.2.2 #1241

#### 11.19.5 Miscellaneous

• docs: update AUTHORS #1222

# 11.20 0.5.6 - 2024-05-23

#### 11.20.1 Added

• feat: add new arc constructors #1208

## **11.20.2 Changed**

• chore: update CHANGELOG for v0.5.5 #1205

# 11.20.3 Dependencies

- build: bump requests from 2.31.0 to 2.32.2 #1204
- build: bump ansys-sphinx-theme[autoapi] from 0.16.0 to 0.16.2 in the docs-deps group #1210
- build: bump docker from 7.0.0 to 7.1.0 #1211
- build: bump scipy from 1.13.0 to 1.13.1 #1212

## 11.21 0.5.5 - 2024-05-21

# 11.21.1 Changed

- docs: adapt ansys\_sphinx\_theme\_autoapi extension for autoapi #1135
- chore: update CHANGELOG for v0.5.4 #1194

#### 11.21.2 Fixed

- fix: adapting Arc class constructor order to (start, end, center) #1196
- chore: limit requests library version under 2.32 #1203

### 11.21.3 Dependencies

- build: bump grpcio from 1.63.0 to 1.64.0 in the grpc-deps group #1198
- build: bump the docs-deps group with 2 updates #1199
- build: bump pytest from 8.2.0 to 8.2.1 #1200

# 11.21.4 Miscellaneous

• chore: pre-commit automatic update #1202

## 11.22 0.5.4 - 2024-05-15

#### 11.22.1 Added

• feat: allow for product\_version on geometry service launcher function #1182

# 11.22.2 Changed

• chore: update CHANGELOG for v0.5.3 #1177

## 11.22.3 Dependencies

- build: bump the docs-deps group with 4 updates #1178
- build: bump pytest from 8.1.1 to 8.2.0 #1179
- build: bump grpcio from 1.62.2 to 1.63.0 in the grpc-deps group #1186
- build: bump the docs-deps group with 2 updates #1187
- build: bump trame-vtk from 2.8.6 to 2.8.7 #1188
- build: bump nbsphinx from 0.9.3 to 0.9.4 in the docs-deps group #1189
- build: bump trame-vtk from 2.8.7 to 2.8.8 #1190

#### 11.22.4 Miscellaneous

- chore: pre-commit automatic update #1180, #1193
- docs: add geometry preparation for Fluent simulation #1183

### 11.23 0.5.3 - 2024-04-29

#### 11.23.1 Fixed

- fix: semver intersphinx mapping not resolved properly #1175
- fix: start and end points for edge #1176

## 11.24 0.5.2 - 2024-04-29

### 11.24.1 Added

• feat: add semver to intersphinx #1173

## 11.24.2 Changed

- chore: update CHANGELOG for v0.5.1 #1165
- chore: bump version to v0.6.dev0 #1166
- chore: update CHANGELOG for v0.5.2 #1172
- fix: allow to reuse last release binaries (if requested) #1174

### 11.24.3 Fixed

• fix: GetSurface and GetCurve not available prior to 24R2 #1171

### 11.24.4 Miscellaneous

- docs: creating a NACA airfoil example #1167
- docs: simplify README example #1169

## 11.25 0.5.1 - 2024-04-24

#### 11.25.1 Added

- feat: security updates dropped for v0.3 or earlier #1126
- feat: add export\_to functions #1147

## 11.25.2 Changed

- ci: adapt to vale v3 #1129
- ci: bump ansys/actions from 5 to 6 in the actions group #1133
- docs: add release notes in our documentation #1138
- chore: bump ansys pre-commit hook to v0.3.0 #1139
- chore: use default vale version #1140
- docs: add user\_agent to Sphinx build #1142
- ci: enabling Linux tests missing #1152
- ci: perform minimal requirements tests #1153

### 11.25.3 Fixed

- fix: docs link in example #1137
- fix: update backend version message #1145
- fix: Trame issues #1148
- fix: Interactive documentation #1160

### 11.25.4 Dependencies

- build: bump ansys-tools-path from 0.5.1 to 0.5.2 #1131
- build: bump the grpc-deps group across 1 directory with 3 updates #1156
- build: bump notebook from 7.1.2 to 7.1.3 in the docs-deps group #1157
- build: bump beartype from 0.18.2 to 0.18.5 #1158

### 11.25.5 Miscellaneous

- docs: add example on exporting designs #1149
- docs: fix link in CHANGELOG.md #1154
- chore: pre-commit automatic update #1159

### 11.26 0.5.0 - 2024-04-17

### 11.26.1 Added

- feat: inserting document into existing design #930
- feat: add changelog action #1023
- feat: create a sphere body on the backend #1035

- feat: mirror a body #1055
- feat: sweeping chains and profiles #1056
- feat: vulnerability checks #1071
- feat: loft profiles #1075
- feat: accept bandit advisories in-line for subprocess #1077
- feat: adding containers to automatic launcher #1090
- feat: minor changes to Linux Dockerfile #1111
- feat: avoid error if folder exists #1125

# 11.26.2 Changed

- build: changing sphinx-autoapi from 3.1.a2 to 3.1.a4 #1038
- chore: add pre-commit.ci configuration #1065
- chore: dependabot PR automatic approval #1067
- ci: bump the actions group with 1 update #1082
- chore: update docker tags to be kept #1085
- chore: update pre-commit versions #1094
- build: use ansys-sphinx-theme autoapi target #1097
- fix: removing @PipKat from \*.md files changelog fragments #1098
- ci: dashboard upload does not apply anymore #1099
- chore: pre-commit.ci not working properly #1108
- chore: update and adding pre-commit.ci config hook #1109
- ci: main Python version update to 3.12 #1112
- ci: skip Linux tests with common approach #1113
- ci: build changelog on release #1118
- chore: update CHANGELOG for v0.5.0 #1119

### 11.26.3 Fixed

- feat: re-enable open file on Linux #817
- fix: adapt export and download tests to new hoops #1057
- fix: linux Dockerfile replace .NET6.0 references by .NET8.0 #1069
- fix: misleading docstring for sweep\_chain() #1070
- fix: prepare\_and\_start\_backend is only available on Windows #1076
- fix: unit tests failing after dms update #1087
- build: beartype upper limit on v0.18 #1095
- fix: improper types being passed for Face and Edge ctor. #1096
- fix: return type should be dict and not ScalarMapContainer (grpc type) #1103
- fix: env version for Dockerfile Windows #1120

- fix: changelog description ill-formatted #1121
- fix: solve issues with intersphinx when releasing #1123

## 11.26.4 Dependencies

- build: bump the docs-deps group with 2 updates #1062, #1093, #1105
- build: bump ansys-api-geometry from 0.3.13 to 0.4.0 #1066
- build: bump the docs-deps group with 1 update #1080
- build: bump pytest-cov from 4.1.0 to 5.0.0 #1081
- build: bump ansys-api-geometry from 0.4.0 to 0.4.1 #1092
- build: bump beartype from 0.17.2 to 0.18.2 #1106
- build: bump ansys-tools-path from 0.4.1 to 0.5.1 #1107
- build: bump panel from 1.4.0 to 1.4.1 in the docs-deps group #1114
- build: bump scipy from 1.12.0 to 1.13.0 #1115

### 11.26.5 Miscellaneous

- [pre-commit.ci] pre-commit autoupdate #1063
- docs: add examples on new methods #1089
- chore: pre-commit automatic update #1116

480

## **PYTHON MODULE INDEX**

```
а
                                              ansys.geometry.core.misc,??
                                              ansys.geometry.core.misc.accuracy,??
ansys.geometry.core,??
                                              ansys.geometry.core.misc.auxiliary,??
ansys.geometry.core.connection,??
                                              ansys.geometry.core.misc.checks,??
ansys.geometry.core.connection.backend,??
                                              ansys.geometry.core.misc.measurements,??
ansys.geometry.core.connection.client,??
                                              ansys.geometry.core.misc.options,??
ansys.geometry.core.connection.conversions,
                                              ansys.geometry.core.misc.units,??
                                              ansys.geometry.core.modeler,??
ansys.geometry.core.connection.defaults,??
ansys.geometry.core.connection.docker_instanceansys.geometry.core.parameters,??
                                              ansys.geometry.core.parameters.parameter,??
                                              ansys.geometry.core.plotting,??
ansys.geometry.core.connection.launcher, ??
ansys. \texttt{geometry.core.connection.product\_instance}, sys. \texttt{geometry.core.plotting.plotter}, \textbf{??}
                                              ansys.geometry.core.plotting.widgets,??
                                              ansys.geometry.core.plotting.widgets.show_design_point,
ansys.geometry.core.connection.validate,??
ansys.geometry.core.designer,??
                                              ansys.geometry.core.shapes,??
ansys.geometry.core.designer.beam,??
                                              ansys.geometry.core.shapes.box_uv, ??
ansys.geometry.core.designer.body, ??
                                              ansys.geometry.core.shapes.curves,??
ansys.geometry.core.designer.component,??
ansys. geometry. core. designer. coordinate\_system \circle, {\tt ??}
                                              ansys.geometry.core.shapes.curves.curve,??
                                              ansys.geometry.core.shapes.curves.curve_evaluation,
ansys.geometry.core.designer.design,??
ansys.geometry.core.designer.designpoint,??
                                              ansys.geometry.core.shapes.curves.ellipse,??
ansys.geometry.core.designer.edge,??
                                              ansys.geometry.core.shapes.curves.line,??
ansys.geometry.core.designer.face,??
                                              sansys.geometry.core.shapes.curves.nurbs, ??
ansys.geometry.core.designer.geometry_command
                                              ansys.geometry.core.shapes.curves.trimmed_curve,
ansys.geometry.core.designer.part, ??
                                              ansys.geometry.core.shapes.parameterization,
ansys.geometry.core.designer.selection,??
ansys.geometry.core.errors,??
                                              ansys.geometry.core.shapes.surfaces,??
ansys.geometry.core.logger,??
                                              ansys.geometry.core.shapes.surfaces.cone, ??
ansys.geometry.core.materials,??
                                              ansys.geometry.core.shapes.surfaces.cylinder,
ansys.geometry.core.materials.material,??
ansys.geometry.core.materials.property,??
                                              ansys.geometry.core.shapes.surfaces.plane,??
ansys.geometry.core.math, ??
                                              ansys.geometry.core.shapes.surfaces.sphere,
ansys.geometry.core.math.bbox, ??
ansys.geometry.core.math.constants,??
                                              ansys.geometry.core.shapes.surfaces.surface,
ansys.geometry.core.math.frame,??
ansys.geometry.core.math.matrix,??
                                              ansys.geometry.core.shapes.surfaces.surface_evaluation,
ansys.geometry.core.math.misc,??
ansys.geometry.core.math.plane,??
                                              ansys.geometry.core.shapes.surfaces.torus,??
ansys.geometry.core.math.point,??
                                              ansys.geometry.core.shapes.surfaces.trimmed_surface,
ansys.geometry.core.math.vector,??
```

```
??
ansys.geometry.core.sketch, ??
ansys.geometry.core.sketch.arc,??
ansys.geometry.core.sketch.box,??
ansys.geometry.core.sketch.circle,??
ansys.geometry.core.sketch.edge,??
ansys.geometry.core.sketch.ellipse,??
ansys.geometry.core.sketch.face,??
ansys.geometry.core.sketch.gears,??
ansys.geometry.core.sketch.polygon, ??
ansys.geometry.core.sketch.segment,??
ansys.geometry.core.sketch.sketch,??
ansys.geometry.core.sketch.slot,??
ansys.geometry.core.sketch.trapezoid,??
ansys.geometry.core.sketch.triangle,??
ansys.geometry.core.tools,??
ansys.geometry.core.tools.check_geometry, ??
ansys.geometry.core.tools.measurement_tools,
ansys.geometry.core.tools.prepare_tools,??
ansys.geometry.core.tools.problem_areas, ??
ansys.geometry.core.tools.repair_tool_message,
ansys.geometry.core.tools.repair_tools,??
ansys.geometry.core.tools.unsupported,??
ansys.geometry.core.typing,??
```

482 Python Module Index