



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

Visualization Interface Tool



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

Jun 19, 2025

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

CONTENTS

1	Getting started	3
1.1	Installation	3
1.1.1	Quick start	3
2	User guide	5
2.1	Default plotter usage	5
2.1.1	Use with PyVista meshes	5
2.1.2	Use with PyAnsys custom objects	5
2.2	Customize your own plotter	6
3	API reference	7
3.1	The <code>ansys.tools.visualization_interface</code> library	7
3.1.1	Summary	7
3.1.2	Description	47
3.1.3	Module detail	47
4	Examples	49
5	Basic usage examples	51
6	Advanced usage examples	53
6.1	Basic usage examples	53
6.1.1	Use trame as a remote service	53
6.1.2	Use a PyVista Qt backend	54
6.1.3	Use a clipping plane	55
6.1.4	Use the <code>MeshObjectPlot</code> class	57
6.1.5	Use the plotter	58
6.1.6	Activate the picker	61
6.2	Advanced usage examples	65
6.2.1	Postprocessing simulation results using the <code>MeshObjectPlot</code> class	66
7	Contribute	69
7.1	Install in developer mode	69
7.2	Run tests	69
7.3	Adhere to code style	69
7.4	Build the documentation	70
7.5	Post issues	70
	Python Module Index	71
	Index	73

The Visualization Interface Tool is a Python API that provides an interface between PyAnsys libraries and different plotting backends.

The Visualization Interface Tool offers these main features:

- Serves as an interface between PyAnsys and other plotting libraries (although only [PyVista](#) is supported currently).
- Provides out-of-the box picking, viewing, and measuring functionalities.
- Supplies an extensible class for adding custom functionalities.

Getting started Learn how to install the Visualization Interface Tool in user mode and quickly begin using it.

Getting started
in your workflow.

User guide Understand key concepts for implementing the Visualization Interface Tool

User guide
Visualization Interface Tool.

API reference Understand how to use Python to interact programmatically with the

API reference
perform many different types of operations.

Examples Explore examples that show how to use the Visualization Interface Tool to

Examples
documentation.

Contribute Learn how to contribute to the Visualization Interface Tool codebase or

Contribute

GETTING STARTED

This section describes how to install the Visualization Interface Tool in user mode and quickly begin using it. If you are interested in contributing to the Visualization Interface Tool, see [Contribute](#) for information on installing in developer mode.

1.1 Installation

To use `pip` to install the Visualization Interface Tool, run this command:

```
pip install ansys-tools-visualization-interface
```

Alternatively, to install the latest version from this library's [GitHub repository](#), run these commands:

```
git clone https://github.com/ansys/ansys-tools-visualization-interface
cd ansys-tools-visualization-interface
pip install .
```

1.1.1 Quick start

The following examples show how to use the Visualization Interface Tool to visualize a mesh file.

This code uses only a PyVista mesh:

```
from ansys.tools.visualization_interface import Plotter

my_mesh = my_custom_object.get_mesh()

# Create a Visualization Interface Tool object
pl = Plotter()
pl.plot(my_mesh)

# Plot the result
pl.show()
```

This code uses objects from a PyAnsys library:

```
from ansys.tools.visualization_interface import Plotter, MeshObjectPlot

my_custom_object = MyObject()
my_mesh = my_custom_object.get_mesh()

mesh_object = MeshObjectPlot(my_custom_object, my_mesh)
```

(continues on next page)

(continued from previous page)

```
# Create a Visualization Interface Tool object
pl = Plotter()
pl.plot(mesh_object)

# Plot the result
pl.show()
```

USER GUIDE

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

2.1 Default plotter usage

The Visualization Interface Tool provides a default plotter that can be used out of the box, using the PyVista backend. This default plotter provides common functionalities so that you do not need to create a custom plotter.

2.1.1 Use with PyVista meshes

You can use the default plotter to plot simple PyVista meshes. This code shows how to use it to visualize a simple PyVista mesh:

```
## Usage example with pyvista meshes ##

import pyvista as pv
from ansys.tools.visualization_interface import Plotter

# Create a pyvista mesh
mesh = pv.Cube()

# Create a plotter
pl = Plotter()

# Add the mesh to the plotter
pl.plot(mesh)

# Show the plotter
pl.show()
```

2.1.2 Use with PyAnsys custom objects

You can also use the default plotter to visualize PyAnsys custom objects. The only requirement is that the custom object must have a method that returns a PyVista mesh a method that exposes a `name` or `id` attribute of your object. To expose a custom object, you use a `MeshObjectPlot` instance. This class relates PyVista meshes with any object.

The following code shows how to use the default plotter to visualize a PyAnsys custom object:

```
## Usage example with PyAnsys custom objects ##
```

(continues on next page)

(continued from previous page)

```
from ansys.tools.visualization_interface import Plotter
from ansys.tools.visualization_interface import MeshObjectPlot

# Create a custom object for this example
class CustomObject:
    def __init__(self):
        self.name = "CustomObject"
        self.mesh = pv.Cube()

    def get_mesh(self):
        return self.mesh

    def name(self):
        return self.name

custom_object = CustomObject()

# Create a MeshObjectPlot instance
mesh_object = MeshObjectPlot(custom_object, custom_object.get_mesh())

# Create a plotter
pl = Plotter()

# Add the MeshObjectPlot instance to the plotter
pl.plot(mesh_object)

# Show the plotter
pl.show()
```

2.2 Customize your own plotter

The Visualization Interface Tool provides a base class, `PlotterInterface`, for customizing certain functions of the plotter. This class provides a set of methods that can be overridden so that you can adapt the plotter to the specific need of your PyAnsys library.

The first thing you must do is to create a class that inherits from the `PlotterInterface` class. After that, see these main use cases for customizing the plotter:

- The most common use case is to customize the way that the objects you represent are shown in the plotter. To this end, you can override the `plot` and `plot_iter` methods. These methods are called every time a new object is added to the plotter. The default implementation of this method is to add a PyVista mesh or a `MeshObjectPlot` instance to the plotter. You can override this method to add your own meshes or objects to the plotter in a manner that fits the way that you want to represent the meshes.
- Another use case is the need to have custom button functionalities for your library. For example, you may want buttons for hiding or showing certain objects. To add custom buttons to the plotter, you use the implementable interface provided by the `PlotterWidget` class.

Some practical examples of how to use the `PlotterInterface` class are included in some PyAnsys libraries, such as `PyAnsys Geometry`.

API REFERENCE

This section describes ansys-tools-visualization-interface endpoints, their capabilities, and how to interact with them programmatically.

3.1 The ansys.tools.visualization_interface library

3.1.1 Summary

Subpackages

<i>backends</i>	Provides interfaces.
<i>types</i>	Provides custom types.
<i>utils</i>	Provides the Utils package.

Submodules

<i>plotter</i>	Module for the Plotter class.
----------------	-------------------------------

Attributes

<code>__version__</code>

Constants

<code>USE_FRAME</code>	
<code>DOCUMENTATION_BUILD</code>	Whether the documentation is being built or not.
<code>TESTING_MODE</code>	Whether the library is being built or not, used to avoid showing plots while testing.
<code>USE_HTML_BACKEND</code>	Whether the library is being built or not, used to avoid showing plots while testing.

The backends package

Summary

Subpackages

<i>pyvista</i>	Provides interfaces.
----------------	----------------------

The pyvista package

Summary

Subpackages

<i>widgets</i>	Provides widgets for the Visualization Interface Tool plotter.
----------------	--

Submodules

<i>pyvista</i>	Provides a wrapper to aid in plotting.
<i>pyvista_interface</i>	Provides plotting for various PyAnsys objects.
<i>trame_local</i>	Provides <i>trame</i> visualizer interface for visualization.
<i>trame_remote</i>	Module for trame websocket client functions.
<i>trame_service</i>	Trame service module.

The widgets package

Summary

Submodules

<i>button</i>	Provides for implementing buttons in PyAnsys.
<i>displace_arrows</i>	Provides the displacement arrows widget for the PyVista plotter.
<i>hide_buttons</i>	Provides the hide buttons widget for the PyAnsys plotter.
<i>measure</i>	Provides the measure widget for the PyAnsys plotter.
<i>mesh_slider</i>	Provides the measure widget for the PyAnsys plotter.
<i>ruler</i>	Provides the ruler widget for the Visualization Interface Tool plotter.
<i>screenshot</i>	Provides the screenshot widget for the Visualization Interface Tool plotter.
<i>view_button</i>	Provides the view button widget for changing the camera view.
<i>widget</i>	Provides the abstract implementation of plotter widgets.

The button.py module

Summary

Classes

<i>Button</i>	Provides the abstract class for implementing buttons in PyAnsys.
---------------	--

Button

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.button.Button(plotter:
    pyvista.Plotter,
    button_config:
    tuple,
    dark_mode:
    bool =
    False)
```

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the abstract class for implementing buttons in PyAnsys.

Parameters

plotter

[Plotter] Plotter to draw the buttons on.

button_config

[tuple] Tuple containing the position and the path to the icon of the button.

dark_mode

[bool, optional] Whether to activate the dark mode or not.

Notes

This class wraps the PyVista `add_checkbox_button_widget()` method.

Overview

Abstract methods

callback Get the functionality of the button, which is implemented by subclasses.

Methods

update Assign the image that represents the button.

Attributes

button_config

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.button import Button
```

Attribute detail

`Button.button_config`

Method detail

abstractmethod `Button.callback(state: bool) → None`

Get the functionality of the button, which is implemented by subclasses.

Parameters

state

[bool] Whether the button is active.

`Button.update()` → `None`

Assign the image that represents the button.

Description

Provides for implementing buttons in PyAnsys.

The `displace_arrows.py` module

Summary

Classes

<code>DisplacementArrow</code>	Defines the arrow to draw and what it is to do.
--------------------------------	---

Enums

<code>CameraPanDirection</code>	Provides an enum with the available movement directions of the camera.
---------------------------------	--

DisplacementArrow

class `ansys.tools.visualization_interface.backends.pyvista.widgets.displace_arrows.DisplacementArrow`(*plotter*, *direction*, *dark_mode*)

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.button.Button`

Defines the arrow to draw and what it is to do.

Parameters

plotter

[Plotter] Plotter to draw the buttons on.

direction

[CameraPanDirection] Direction that the camera is to move.

dark_mode

[bool, optional] Whether to activate the dark mode or not.

Overview

Methods

<i>callback</i>	Move the camera in the direction defined by the button.
-----------------	---

Attributes

<i>direction</i>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.displace_arrows import DisplacementArrow
```

Attribute detail

DisplacementArrow.**direction**

Method detail

DisplacementArrow.**callback**(*state: bool*) → None

Move the camera in the direction defined by the button.

Parameters

state

[bool] Whether the state of the button, which is inherited from PyVista, is active. However, this parameter is unused by this callback method.

CameraPanDirection

class ansys.tools.visualization_interface.backends.pyvista.widgets.displace_arrows.CameraPanDirection(*args, **kwargs)

Bases: `enum.Enum`

Provides an enum with the available movement directions of the camera.

Overview

Attributes

<i>XUP</i>
<i>XDOWN</i>
<i>YUP</i>
<i>YDOWN</i>
<i>ZUP</i>
<i>ZDOWN</i>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.displace_arrows import CameraPanDirection
```

Attribute detail

```
CameraPanDirection.XUP = (0, 'upxarrow', (5, 170))
CameraPanDirection.XDOWN = (1, 'downarrow', (5, 130))
CameraPanDirection.YUP = (2, 'upyarrow', (35, 170))
CameraPanDirection.YDOWN = (3, 'downarrow', (35, 130))
CameraPanDirection.ZUP = (4, 'upzarrow', (65, 170))
CameraPanDirection.ZDOWN = (5, 'downarrow', (65, 130))
```

Description

Provides the displacement arrows widget for the PyVista plotter.

The `hide_buttons.py` module

Summary

Classes

<i>HideButton</i>	Provides the hide widget for the Visualization Interface Tool Plotter class.
-------------------	--

HideButton

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.hide_buttons.HideButton(plotter:
                                                                 an-
                                                                 sys.tools.visuali
                                                                 dark_mode:
                                                                 bool
                                                                 =
                                                                 False)
```

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the hide widget for the Visualization Interface Tool Plotter class.

Parameters

plotter_helper

[PlotterHelper] Plotter to add the hide widget to.

dark_mode

[bool, optional] Whether to activate the dark mode or not.

Overview

Methods

<code>callback</code>	Remove or add the hide widget actor upon click.
<code>update</code>	Define the hide widget button parameters.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.hide_buttons import HideButton
```

Method detail

`HideButton.callback(state: bool) → None`

Remove or add the hide widget actor upon click.

Parameters

state

[bool] Whether the state of the button, which is inherited from PyVista, is active.

`HideButton.update() → None`

Define the hide widget button parameters.

Description

Provides the hide buttons widget for the PyAnsys plotter.

The `measure.py` module

Summary

Classes

<code>MeasureWidget</code>	Provides the measure widget for the Visualization Interface Tool Plotter class.
----------------------------	---

MeasureWidget

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.measure.MeasureWidget(plotter_helper:
    an-
    sys.tools.visualization_interface.backends.pyvista.widgets.measure.
    dark_mode:
    bool
    =
    False)
```

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the measure widget for the Visualization Interface Tool Plotter class.

Parameters

plotter_helper

[PlotterHelper] Plotter to add the measure widget to.

dark_mode

[bool, optional] Whether to activate the dark mode or not.

Overview**Methods**

<i>callback</i>	Remove or add the measurement widget actor upon click.
<i>update</i>	Define the measurement widget button parameters.

Attributes

<i>plotter_helper</i>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.measure import   
↳ MeasureWidget
```

Attribute detail

MeasureWidget.**plotter_helper**

Method detail

MeasureWidget.**callback**(state: bool) → None

Remove or add the measurement widget actor upon click.

Parameters**state**

[bool] Whether the state of the button, which is inherited from PyVista, is active.

MeasureWidget.**update**() → None

Define the measurement widget button parameters.

Description

Provides the measure widget for the PyAnsys plotter.

The mesh_slider.py module**Summary****Classes**

<i>MeshSliderWidget</i>	Provides the mesh slider widget for the Visualization Interface Tool Plotter class.
-------------------------	---

MeshSliderWidget

`class ansys.tools.visualization_interface.backends.pyvista.widgets.mesh_slider.MeshSliderWidget`*(plotter_helper: ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget, dark_mode: bool = False)*

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the mesh slider widget for the Visualization Interface Tool Plotter class.

Parameters

- plotter_helper**
[PlotterHelper] Plotter to add the mesh slider widget to.
- dark_mode**
[bool, optional] Whether to activate the dark mode or not.

Overview

Methods

<code>callback</code>	Remove or add the mesh slider widget actor upon click.
<code>update</code>	Define the mesh slider widget button parameters.

Attributes

`plotter_helper`

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.mesh_slider import MeshSliderWidget
```

Attribute detail

`MeshSliderWidget.plotter_helper`

Method detail

`MeshSliderWidget.callback`*(state: bool) → None*
Remove or add the mesh slider widget actor upon click.

Parameters

- state**
[bool] Whether the state of the button, which is inherited from PyVista, is active.

`MeshSliderWidget.update`*() → None*
Define the mesh slider widget button parameters.

Description

Provides the measure widget for the PyAnsys plotter.

The ruler.py module

Summary

Classes

<i>Ruler</i>	Provides the ruler widget for the Visualization Interface Tool <code>Plotter</code> class.
--------------	--

Ruler

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.ruler.Ruler(plotter:
                                                                              pyvista.Plotter,
                                                                              dark_mode:
                                                                              bool =
                                                                              False)
```

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the ruler widget for the Visualization Interface Tool `Plotter` class.

Parameters

plotter

[`Plotter`] Provides the plotter to add the ruler widget to.

dark_mode

[`bool`, optional] Whether to activate the dark mode or not.

Overview

Methods

<i>callback</i>	Remove or add the ruler widget actor upon click.
<i>update</i>	Define the configuration and representation of the ruler widget button.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.ruler import Ruler
```

Method detail

`Ruler.callback(state: bool)` → `None`

Remove or add the ruler widget actor upon click.

Parameters

state

[`bool`] Whether the state of the button, which is inherited from `PyVista`, is `True`.

Notes

This method provides a callback function for the ruler widget. It is called every time the ruler widget is clicked.

Ruler.**update()** → `None`

Define the configuration and representation of the ruler widget button.

Description

Provides the ruler widget for the Visualization Interface Tool plotter.

The screenshot.py module

Summary

Classes

<i>ScreenshotButton</i>	Provides the screenshot widget for the Visualization Interface Tool Plotter class.
-------------------------	--

ScreenshotButton

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.screenshot.ScreenshotButton(plotter:
    pyvista.Plotter,
    dark_mode:
        bool
        =
        False)
```

Bases: `ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget`

Provides the screenshot widget for the Visualization Interface Tool Plotter class.

Parameters

plotter
`[Plotter]` Provides the plotter to add the screenshot widget to.

dark_mode
`[bool, optional]` Whether to activate the dark mode or not.

Overview

Methods

<i>callback</i>	Remove or add the screenshot widget actor upon click.
<i>update</i>	Define the configuration and representation of the screenshot widget button.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.screenshot import
    ScreenshotButton
```

Method detail

ScreenshotButton.**callback**(*state: bool*) → None

Remove or add the screenshot widget actor upon click.

Parameters

state

[bool] Whether the state of the button, which is inherited from PyVista, is True.

Notes

This method provides a callback function for the screenshot widget. It is called every time the screenshot widget is clicked.

ScreenshotButton.**update**() → None

Define the configuration and representation of the screenshot widget button.

Description

Provides the screenshot widget for the Visualization Interface Tool plotter.

The view_button.py module

Summary

Classes

<i>ViewButton</i>	Provides for changing the view.
-------------------	---------------------------------

Enums

<i>ViewDirection</i>	Provides an enum with the available views.
----------------------	--

ViewButton

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.view_button.ViewButton(plotter:
    pyvista.Plotter,
    direction:
    tuple,
    dark_mode:
    bool
    =
    False)
```

Bases: *ansys.tools.visualization_interface.backends.pyvista.widgets.button.Button*

Provides for changing the view.

Parameters

plotter

[Plotter] Plotter to draw the buttons on.

direction

[ViewDirection] Direction of the view.

dark_mode

[bool, optional] Whether to activate the dark mode or not.

Overview**Methods**

<i>callback</i>	Change the view depending on button interaction.
-----------------	--

Attributes

<i>direction</i>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.view_button import ViewButton
```

Attribute detail

ViewButton.direction

Method detailViewButton.callback(*state: bool*) → None

Change the view depending on button interaction.

Parameters**state**

[bool] Whether the state of the button, which is inherited from PyVista, is True.

Raises**NotImplementedError**

Raised if the specified direction is not implemented.

ViewDirection

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.view_button.ViewDirection(*args, **kwargs)
```

Bases: `enum.Enum`

Provides an enum with the available views.

Overview

Attributes

<code>XYPLUS</code>
<code>XYMINUS</code>
<code>XZPLUS</code>
<code>XZMINUS</code>
<code>YZPLUS</code>
<code>YZMINUS</code>
<code>ISOMETRIC</code>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.view_button import ↪ViewDirection
```

Attribute detail

```
ViewDirection.XYPLUS = (0, '+xy', (5, 220))
ViewDirection.XYMINUS = (1, '-xy', (5, 251))
ViewDirection.XZPLUS = (2, '+xz', (5, 282))
ViewDirection.XZMINUS = (3, '-xz', (5, 313))
ViewDirection.YZPLUS = (4, '+yz', (5, 344))
ViewDirection.YZMINUS = (5, '-yz', (5, 375))
ViewDirection.ISOMETRIC = (6, 'isometric', (5, 406))
```

Description

Provides the view button widget for changing the camera view.

The widget.py module

Summary

Classes

<code>PlotterWidget</code>	Provides an abstract class for plotter widgets.
----------------------------	---

PlotterWidget

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget(plotter: pyvista.Plotter)
```

Bases: `abc.ABC`

Provides an abstract class for plotter widgets.

Parameters

plotter

[`Plotter`] Plotter instance to add the widget to.

Notes

These widgets are intended to be used with PyVista plotter objects. More specifically, the way in which this abstraction has been built ensures that these widgets can be easily integrated with the Visualization Interface Tool's widgets.

Overview**Abstract methods**

<code>callback</code>	General callback function for <code>PlotterWidget</code> objects.
<code>update</code>	General update function for <code>PlotterWidget</code> objects.

Properties

<code>plotter</code>	Plotter object that the widget is assigned to.
----------------------	--

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.widget import PlotterWidget
```

Property detail

property `PlotterWidget.plotter`: `pyvista.Plotter`

Plotter object that the widget is assigned to.

Method detail

abstractmethod `PlotterWidget.callback(state)` → `None`

General callback function for `PlotterWidget` objects.

abstractmethod `PlotterWidget.update()` → `None`

General update function for `PlotterWidget` objects.

Description

Provides the abstract implementation of plotter widgets.

Description

Provides widgets for the Visualization Interface Tool plotter.

The `pyvista.py` module**Summary**

Classes

<i>PyVistaBackendInterface</i>	Provides the interface for the Visualization Interface Tool plotter.
<i>PyVistaBackend</i>	Provides the generic plotter implementation for PyAnsys libraries.

Constants

<i>DARK_MODE_THRESHOLD</i>

PyVistaBackendInterface

```

class ansys.tools.visualization_interface.backends.pyvista.pyvista.PyVistaBackendInterface(
    use_trame: bool
    |
    None
    =
    None,
    allow_picking: bool
    |
    None
    =
    False,
    allow_hovering: bool
    |
    None
    =
    False,
    plot_picked_name: bool
    |
    None
    =
    False,
    show_plane: bool
    |
    None
    =
    False,
    use_qt: bool
    |
    None
    =
    False,
    show_qt: bool
    |
    None
    =
    True,
    **plotter_kwargs)

```

Bases: `ansys.tools.visualization_interface.backends._base.BaseBackend`

Provides the interface for the Visualization Interface Tool plotter.

This class is intended to be used as a base class for the custom plotters in the different PyAnsys libraries. It provides the basic plotter functionalities, such as adding objects and enabling widgets and picking capabilities. It also provides the ability to show the plotter using the `trame` service.

You can override the `plot_iter()`, `plot()`, and `picked_operation()` methods. The `plot_iter()` method

is intended to plot a list of objects to the plotter, while the `plot()` method is intended to plot a single object to the plotter. The `show()` method is intended to show the plotter. The `picked_operation()` method is intended to perform an operation on the picked objects.

Parameters

- use_frame**
[Optional[bool], default: `None`] Whether to activate the usage of the frame UI instead of the Python window.
- allow_picking**
[Optional[bool], default: `False`] Whether to allow picking capabilities in the window. Incompatible with hovering. Picking will take precedence over hovering.
- allow_hovering**
[Optional[bool], default: `False`] Whether to allow hovering capabilities in the window. Incompatible with picking. Picking will take precedence over hovering.
- plot_picked_names**
[Optional[bool], default: `False`] Whether to plot the names of the picked objects.
- show_plane**
[Optional[bool], default: `False`] Whether to show the plane in the plotter.
- use_qt**
[Optional[bool], default: `False`] Whether to use the Qt backend for the plotter.
- show_qt**
[Optional[bool], default: `True`] Whether to show the Qt window.

Overview

Abstract methods

<code>plot_iter</code>	Plot one or more compatible objects to the plotter.
<code>plot</code>	Plot a single object to the plotter.

Methods

<code>enable_widgets</code>	Enable the widgets for the plotter.
<code>add_widget</code>	Add one or more custom widgets to the plotter.
<code>select_object</code>	Select a custom object in the plotter.
<code>unselect_object</code>	Unselect a custom object in the plotter.
<code>picker_callback</code>	Define the callback for the element picker.
<code>hover_callback</code>	Define the callback for the element hover.
<code>compute_edge_object_map</code>	Compute the mapping between plotter actors and EdgePlot objects.
<code>enable_picking</code>	Enable picking capabilities in the plotter.
<code>enable_hover</code>	Enable hover capabilities in the plotter.
<code>disable_picking</code>	Disable picking capabilities in the plotter.
<code>disable_hover</code>	Disable hover capabilities in the plotter.
<code>show</code>	Plot and show any PyAnsys object.
<code>show_plotter</code>	Show the plotter or start the <code>frame</code> service.
<code>picked_operation</code>	Perform an operation on the picked objects.

Properties

<code>pv_interface</code>	PyVista interface.
<code>scene</code>	PyVista scene.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.pyvista import _  
↳ PyVistaBackendInterface
```

Property detail

property `PyVistaBackendInterface.pv_interface:`
`ansys.tools.visualization_interface.backends.pyvista.pyvista_interface.PyVistaInterface`
 PyVista interface.

property `PyVistaBackendInterface.scene:` `pyvista.Plotter`
 PyVista scene.

Method detail

`PyVistaBackendInterface.enable_widgets(dark_mode: bool = False) → None`

Enable the widgets for the plotter.

Parameters

dark_mode
`[bool, default: False]` Whether to use dark mode for the widgets.

`PyVistaBackendInterface.add_widget(widget: an-
 sys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget
 |
 List[ansys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget])`

Add one or more custom widgets to the plotter.

Parameters

widget
`[Union[PlotterWidget, List[PlotterWidget]]]` One or more custom widgets.

`PyVistaBackendInterface.select_object(custom_object: an-
 sys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot
 | ansys.tools.visualization_interface.types.edge_plot.EdgePlot, pt:
 numpy.ndarray) → None`

Select a custom object in the plotter.

This method highlights the edges of a body and adds a label. It also adds the object to the `_picked_dict` and the actor to the `_picker_added_actors_map`.

Parameters

custom_object
`[Union[MeshObjectPlot, EdgePlot]]` Custom object to select.

pt
`[ndarray]` Set of points to determine the label position.

`PyVistaBackendInterface.unselect_object(custom_object: ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot | ansys.tools.visualization_interface.types.edge_plot.EdgePlot) → None`

Unselect a custom object in the plotter.

This method removes edge highlighting and the label from a plotter actor and removes the object from the Visualization Interface Tool object selection.

Parameters

`custom_object`

`[Union[MeshObjectPlot, EdgePlot]]` Custom object to unselect.

`PyVistaBackendInterface.picker_callback(actor: pyvista.Actor) → None`

Define the callback for the element picker.

Parameters

`actor`

`[Actor]` Actor to select for the picker.

`PyVistaBackendInterface.hover_callback(_widget, event_name) → None`

Define the callback for the element hover.

Parameters

`actor`

`[Actor]` Actor to hover for the picker.

`PyVistaBackendInterface.compute_edge_object_map() → Dict[pyvista.Actor, ansys.tools.visualization_interface.types.edge_plot.EdgePlot]`

Compute the mapping between plotter actors and EdgePlot objects.

Returns

`Dict[Actor, EdgePlot]`

Dictionary defining the mapping between plotter actors and EdgePlot objects.

`PyVistaBackendInterface.enable_picking()`

Enable picking capabilities in the plotter.

`PyVistaBackendInterface.enable_hover()`

Enable hover capabilities in the plotter.

`PyVistaBackendInterface.disable_picking()`

Disable picking capabilities in the plotter.

`PyVistaBackendInterface.disable_hover()`

Disable hover capabilities in the plotter.

`PyVistaBackendInterface.show(plottable_object: Any = None, screenshot: str | None = None, view_2d: Dict = None, name_filter: str = None, dark_mode: bool = False, plotting_options: Dict[str, Any] | None = {}, **show_options: Dict[str, Any]) → List[Any]`

Plot and show any PyAnsys object.

The types of objects supported are MeshObjectPlot, pv.MultiBlock, and pv.PolyData.

Parameters

`plottable_object`

`[Any, default: None]` Object or list of objects to plot.

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

view_2d
 [`Dict`, default: `None`] Dictionary with the plane and the viewup vectors of the 2D plane.

name_filter
 [`str`, default: `None`] Regular expression with the desired name or names to include in the plotter.

dark_mode
 [`bool`, default: `False`] Whether to use dark mode for the widgets.

plotting_options
 [`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

****show_options**
 [`Any`] Additional keyword arguments for the show method.

Returns

List[`Any`]
 List with the picked bodies in the picked order.

`PyVistaBackendInterface.show_plotter(screenshot: str | None = None, **kwargs) → None`

Show the plotter or start the `trame` service.

Parameters

plotter
 [`Plotter`] Visualization Interface Tool plotter with the meshes added.

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

abstractmethod `PyVistaBackendInterface.plot_iter(plottable_object: Any, name_filter: str = None, **plotting_options)`

Plot one or more compatible objects to the plotter.

Parameters

plottable_object
 [`Any`] One or more objects to add.

name_filter
 [`str`, default: `None`.] Regular expression with the desired name or names to include in the plotter.

****plotting_options**
 [`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

abstractmethod `PyVistaBackendInterface.plot(plottable_object: Any, name_filter: str = None, **plotting_options)`

Plot a single object to the plotter.

Parameters

plottable_object
 [`Any`] Object to add.

name_filter

[*str*] Regular expression with the desired name or names to include in the plotter.

****plotting_options**

[*dict*, default: *None*] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

`PyVistaBackendInterface.picked_operation()` → *None*

Perform an operation on the picked objects.

PyVistaBackend

```
class ansys.tools.visualization_interface.backends.pyvista.pyvista.PyVistaBackend(use_frame:
    bool |
    None =
    None, allow_picking:
    bool |
    None =
    False, allow_hovering:
    bool |
    None =
    False,
    plot_picked_names:
    bool |
    None =
    True,
    use_qt:
    bool |
    None =
    False,
    show_qt:
    bool |
    None =
    False)
```

Bases: `PyVistaBackendInterface`

Provides the generic plotter implementation for PyAnsys libraries.

This class accepts `MeshObjectPlot`, `pv.MultiBlock` and `pv.PolyData` objects.

Parameters**use_frame**

[*bool*, default: *None*] Whether to enable the use of `frame`. The default is *None*, in which case the `USE_FRAME` global setting is used.

allow_picking

[*Optional[bool]*, default: *False*] Whether to allow picking capabilities in the window. Incompatible with hovering. Picking will take precedence over hovering.

allow_hovering

[*Optional[bool]*, default: *False*] Whether to allow hovering capabilities in the window. Incompatible with picking. Picking will take precedence over hovering.

plot_picked_names

[*bool*, default: *True*] Whether to plot the names of the picked objects.

Overview

Methods

<code>plot_iter</code>	Plot the elements of an iterable of any type of object to the scene.
<code>plot</code>	Plot a pyansys or PyVista object to the plotter.
<code>close</code>	Close the plotter for PyVistaQT.

Properties

<code>base_plotter</code>	Return the base plotter object.
---------------------------	---------------------------------

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.pyvista import PyVistaBackend
```

Property detail

property `PyVistaBackend.base_plotter`

Return the base plotter object.

Method detail

`PyVistaBackend.plot_iter(plotting_list: List[Any], name_filter: str = None, **plotting_options) → None`

Plot the elements of an iterable of any type of object to the scene.

The types of objects supported are `Body`, `Component`, `List[pv.PolyData]`, `pv.MultiBlock`, and `Sketch`.

Parameters

plotting_list

`[List[Any]]` List of objects to plot.

name_filter

`[str, default: None]` Regular expression with the desired name or names to include in the plotter.

****plotting_options**

`[dict, default: None]` Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

`PyVistaBackend.plot(plottable_object: Any, name_filter: str = None, **plotting_options)`

Plot a pyansys or PyVista object to the plotter.

Parameters

plottable_object

`[Any]` Object to add.

name_filter

`[str]` Regular expression with the desired name or names to include in the plotter.

****plotting_options**

`[dict, default: None]` Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

`PyVistaBackend.close()`

Close the plotter for PyVistaQT.

Description

Provides a wrapper to aid in plotting.

Module detail

`pyvista.DARK_MODE_THRESHOLD = 120`

The `pyvista_interface.py` module

Summary

Classes

<i>PyVistaInterface</i>	Provides the middle class between PyVista plotting operations and PyAnsys objects.
-------------------------	--

PyVistaInterface

```

class ansys.tools.visualization_interface.backends.pyvista.pyvista_interface.PyVistaInterface(scene:
    pyvista.Plotter
    |
    None
    =
    None,
    color_opts:
    Dict
    |
    None
    =
    None,
    num_points:
    int
    =
    100,
    enable_widgets:
    bool
    =
    True,
    show_plane:
    bool
    =
    False,
    use_qt:
    bool
    =
    False,
    show_qt:
    bool
    =
    True,
    **plotter_kwargs)

```

Provides the middle class between PyVista plotting operations and PyAnsys objects.

The main purpose of this class is to simplify interaction between PyVista and the PyVista backend provided. This class is responsible for creating the PyVista scene and adding the PyAnsys objects to it.

Parameters

scene

[[Plotter](#), default: [None](#)] Scene for rendering the objects. If passed, `off_screen` needs to be set manually beforehand for documentation and testing.

color_opts

[[dict](#), default: [None](#)] Dictionary containing the background and top colors.

num_points

[[int](#), default: 100] Number of points to use to render the shapes.

enable_widgets

[[bool](#), default: [True](#)] Whether to enable widget buttons in the plotter window. Widget buttons must be disabled when using [trame](#) for visualization.

show_plane

[bool, default: `False`] Whether to show the XY plane in the plotter window.

use_qt

[bool, default: `False`] Whether to use the Qt backend for the plotter window.

show_qt

[bool, default: `True`] Whether to show the Qt plotter window.

Overview

Methods

<code>view_xy</code>	View the scene from the XY plane.
<code>view_xz</code>	View the scene from the XZ plane.
<code>view_yx</code>	View the scene from the YX plane.
<code>view_yz</code>	View the scene from the YZ plane.
<code>view_zx</code>	View the scene from the ZX plane.
<code>view_zy</code>	View the scene from the ZY plane.
<code>clip</code>	Clip a given mesh with a plane.
<code>plot_meshobject</code>	Plot a generic <code>MeshObjectPlot</code> object to the scene.
<code>plot_edges</code>	Plot the outer edges of an object to the plot.
<code>plot</code>	Plot any type of object to the scene.
<code>plot_iter</code>	Plot elements of an iterable of any type of objects to the scene.
<code>show</code>	Show the rendered scene on the screen.
<code>set_add_mesh_defaults</code>	Set the default values for the plotting options.

Properties

<code>scene</code>	Rendered scene object.
<code>object_to_actors_map</code>	Mapping between the PyVista actor and the PyAnsys objects.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.pyvista_interface import_  
↳ PyVistaInterface
```

Property detail

property `PyVistaInterface.scene`: `pyvista.plotting.plotter.Plotter`

Rendered scene object.

Returns

`Plotter`

Rendered scene object.

property `PyVistaInterface.object_to_actors_map`: `Dict[pyvista.Actor, ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot]`

Mapping between the PyVista actor and the PyAnsys objects.

Method detail

`PyVistaInterface.view_xy()` → `None`

View the scene from the XY plane.

`PyVistaInterface.view_xz()` → `None`

View the scene from the XZ plane.

`PyVistaInterface.view_yx()` → `None`

View the scene from the YX plane.

`PyVistaInterface.view_yz()` → `None`

View the scene from the YZ plane.

`PyVistaInterface.view_zx()` → `None`

View the scene from the ZX plane.

`PyVistaInterface.view_zy()` → `None`

View the scene from the ZY plane.

`PyVistaInterface.clip(mesh: pyvista.PolyData | pyvista.MultiBlock | pyvista.UnstructuredGrid, plane: ansys.tools.visualization_interface.utils.clip_plane.ClipPlane)` → `pyvista.PolyData` | `pyvista.MultiBlock`

Clip a given mesh with a plane.

Parameters

mesh

[`Union`[`pv.PolyData`, `pv.MultiBlock`]] Mesh.

normal

[`str`, default: "x"] Plane to use for clipping. Options are "x", "-x", "y", "-y", "z", and "-z".

origin

[`tuple`, default: `None`] Origin point of the plane.

plane

[`ClipPlane`, default: `None`] Clipping plane to cut the mesh with.

Returns

`Union`[`pv.PolyData`, `pv.MultiBlock`]

Clipped mesh.

`PyVistaInterface.plot_meshobject(custom_object: ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot, **plotting_options)`

Plot a generic `MeshObjectPlot` object to the scene.

Parameters

plottable_object

[`MeshObjectPlot`] Object to add to the scene.

****plotting_options**

[`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

```
PyVistaInterface.plot_edges(custom_object:  
    ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot,  
    **plotting_options) → None
```

Plot the outer edges of an object to the plot.

This method has the side effect of adding the edges to the `MeshObjectPlot` object that you pass through the parameters.

Parameters

custom_object

[`MeshObjectPlot`] Custom object with the edges to add.

****plotting_options**

[`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

```
PyVistaInterface.plot(plottable_object: pyvista.PolyData | pyvista.MultiBlock |  
    ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot |  
    pyvista.UnstructuredGrid, name_filter: str = None, **plotting_options) → None
```

Plot any type of object to the scene.

Supported object types are `List[pv.PolyData]`, `MeshObjectPlot`, and `pv.MultiBlock`.

Parameters

plottable_object

[`Union[pv.PolyData, pv.MultiBlock, MeshObjectPlot, pv.UnstructuredGrid, pv.StructuredGrid]`] Object to plot.

name_filter

[`str`, default: `None`] Regular expression with the desired name or names to include in the plotter.

****plotting_options**

[`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

```
PyVistaInterface.plot_iter(plotting_list: List[Any], name_filter: str = None, **plotting_options) → None
```

Plot elements of an iterable of any type of objects to the scene.

Parameters

plotting_list

[`List[Any]`] List of objects to plot.

name_filter

[`str`, default: `None`] Regular expression with the desired name or names to include in the plotter.

****plotting_options**

[`dict`, default: `None`] Keyword arguments. For allowable keyword arguments, see the `Plotter.add_mesh` method.

```
PyVistaInterface.show(show_plane: bool = False, jupyter_backend: str | None = None, **kwargs: Dict |  
    None) → None
```

Show the rendered scene on the screen.

Parameters

show_plane

[`bool`, default: `True`] Whether to show the XY plane.

jupyter_backend

[[str](#), default: [None](#)] PyVista Jupyter backend.

****kwargs**

[[dict](#), default: [None](#)] Plotting keyword arguments. For allowable keyword arguments, see the [Plotter.show](#) method.

Notes

For more information on supported Jupyter backends, see [Jupyter Notebook Plotting](#) in the PyVista documentation.

`PyVistaInterface.set_add_mesh_defaults(plotting_options: Dict | None) → None`

Set the default values for the plotting options.

Parameters**plotting_options**

[Optional[Dict]] Keyword arguments. For allowable keyword arguments, see the [Plotter.add_mesh](#) method.

Description

Provides plotting for various PyAnsys objects.

The trame_local.py module**Summary****Classes**

<i>TrameVisualizer</i>	Defines the trame layout view.
--	--------------------------------

Constants

<i>CLIENT_TYPE</i>

TrameVisualizer

class `ansys.tools.visualization_interface.backends.pyvista.trame_local.TrameVisualizer`

Defines the trame layout view.

Overview**Methods**

<i>set_scene</i>	Set the trame layout view and the mesh to show through the PyVista plotter.
<i>show</i>	Start the trame server and show the mesh.

Attributes

<code>server</code>

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.trame_local import _
↳ TrameVisualizer
```

Attribute detail

`TrameVisualizer.server = None`

Method detail

`TrameVisualizer.set_scene(plotter)`

Set the trame layout view and the mesh to show through the PyVista plotter.

Parameters

plotter

[`Plotter`] PyVista plotter with the rendered mesh.

`TrameVisualizer.show()`

Start the trame server and show the mesh.

Description

Provides `trame` visualizer interface for visualization.

Module detail

`trame_local.CLIENT_TYPE = 'vue2'`

The `trame_remote.py` module

Summary

Functions

<code>send_pl</code>	Send the plotter meshes to a remote trame service.
<code>send_mesh</code>	Send a mesh to a remote trame service.

Description

Module for trame websocket client functions.

Module detail

```
trame_remote.send_pl(plotter: pyvista.Plotter, host: str = 'localhost', port: int = 8765)
```

Send the plotter meshes to a remote trame service.

Since plotter can't be pickled, we send the meshes list instead.

Parameters

plotter

[*pv.Plotter*] Plotter to send.

host

[*str*, optional] Websocket host to connect to, by default “localhost”.

port

[*int*, optional] Websocket port to connect to, by default 8765.

```
trame_remote.send_mesh(mesh: pyvista.PolyData | pyvista.MultiBlock, host: str = 'localhost', port: int = 8765)
```

Send a mesh to a remote trame service.

Parameters

mesh

[Union[*pv.PolyData*, *pv.MultiBlock*]] Mesh to send.

host

[*str*, optional] Websocket host to connect to, by default “localhost”.

port

[*int*, optional] Websocket port to connect to, by default 8765.

The trame_service.py module

Summary

Classes

<i>TrameService</i>	Trame service class.
---------------------	----------------------

TrameService

```
class ansys.tools.visualization_interface.backends.pyvista.trame_service.TrameService(websocket_host:
```

```
str
=
'lo-
cal-
host',
web-
socket_port:
int
=
8765)
```

Trame service class.

Initializes a trame service where you can send meshes to plot in a trame webview plotter.

Parameters

websocket_host

[[str](#), optional] Host where the webserver will listen for new plotters and meshes, by default “localhost”.

websocket_port

[[int](#), optional] Port where the webserver will listen for new plotters and meshes, by default 8765.

Overview**Methods**

clear_plotter	Clears the web view in the service.
set_scene	Sets the web view scene for the frame service.
run	Start the frame web view and the websocket services.

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.frame_service import FrameService
```

Method detail**FrameService.clear_plotter()**

Clears the web view in the service.

FrameService.set_scene()

Sets the web view scene for the frame service.

FrameService.run()

Start the frame web view and the websocket services.

Description

Frame service module.

Description

Provides interfaces.

Description

Provides interfaces.

The types package**Summary****Submodules**

edge_plot	Provides the edge type for plotting.
mesh_object_plot	Provides the MeshObjectPlot class.

The edge_plot.py module

Summary

Classes

EdgePlot	Provides the mapper class for relating PyAnsys object edges with its PyVista actor.
-----------------	---

EdgePlot

class ansys.tools.visualization_interface.types.edge_plot.**EdgePlot**(actor: *pyvista.Actor*,
edge_object: *Any*, parent:
Any = None)

Provides the mapper class for relating PyAnsys object edges with its PyVista actor.

Parameters

actor

[**Actor**] PyVista actor that represents the edge.

edge_object

[Edge] PyAnsys object edge that is represented by the PyVista actor.

parent

[MeshObjectPlot, default: **None**] Parent PyAnsys object of the edge.

Overview

Properties

<i>actor</i>	PyVista actor of the object.
<i>edge_object</i>	PyAnsys edge.
<i>parent</i>	Parent PyAnsys object of the edge.
<i>name</i>	Name of the edge.

Import detail

```
from ansys.tools.visualization_interface.types.edge_plot import EdgePlot
```

Property detail

property EdgePlot.actor: **pyvista.Actor**

PyVista actor of the object.

Returns

Actor

PyVista actor.

property EdgePlot.edge_object: **Any**

PyAnsys edge.

Returns

Any

PyAnsys edge.

property EdgePlot.parent: **Any**

Parent PyAnsys object of the edge.

Returns

Any

Parent PyAnsys object.

property EdgePlot.name: **str**

Name of the edge.

Returns

str

Name of the edge.

Description

Provides the edge type for plotting.

The mesh_object_plot.py module

Summary

Classes

<i>MeshObjectPlot</i>	Relates a custom object with a mesh, provided by the consumer library.
-----------------------	--

MeshObjectPlot

```
class ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot(custom_object: Any, mesh: pyvista.PolyData | pyvista.MultiBlock, actor: pyvista.Actor = None, edges: List[ansys.tools.visualization_ = None])
```

Relates a custom object with a mesh, provided by the consumer library.

Overview

Properties

<i>mesh</i>	Mesh of the object in PyVista format.
<i>custom_object</i>	Custom object.
<i>actor</i>	PyVista actor of the object in the plotter.
<i>edges</i>	Edges of the object.
<i>name</i>	Name of the object.

Import detail

```
from ansys.tools.visualization_interface.types.mesh_object_plot import MeshObjectPlot
```

Property detail

property MeshObjectPlot.mesh: `pyvista.PolyData` | `pyvista.MultiBlock`

Mesh of the object in PyVista format.

Returns

`Union[pv.PolyData, pv.MultiBlock]`

Mesh of the object.

property MeshObjectPlot.custom_object: `Any`

Custom object.

Returns

`Any`

Custom object.

property MeshObjectPlot.actor: `pyvista.Actor`

PyVista actor of the object in the plotter.

Returns

`pv.Actor`

PyVista actor of the object.

property MeshObjectPlot.edges:

`List[ansys.tools.visualization_interface.types.edge_plot.EdgePlot]`

Edges of the object.

Returns

`List[EdgePlot]`

Edges of the object.

property MeshObjectPlot.name: `str`

Name of the object.

Returns

`str`

Name of the object.

Description

Provides the MeshObjectPlot class.

Description

Provides custom types.

The utils package

Summary

Submodules

<code>clip_plane</code>	Provides the ClipPlane class.
<code>color</code>	Provides an enum with the color to use for the plotter actors.
<code>logger</code>	Provides the singleton helper class for the logger.

The clip_plane.py module

Summary

Classes

<code>ClipPlane</code>	Provides the clipping plane for clipping meshes in the plotter.
------------------------	---

ClipPlane

class ansys.tools.visualization_interface.utils.clip_plane.**ClipPlane**(*normal: Tuple[float, float, float] = (1, 0, 0), origin: Tuple[float, float, float] = (0, 0, 0)*)

Provides the clipping plane for clipping meshes in the plotter.

The clipping plane is defined by both normal and origin vectors.

Parameters

normal

[Tuple[float, float, float], default: (1, 0, 0)] Normal of the plane.

origin

[Tuple[float, float, float], default: (0, 0, 0)] Origin point of the plane.

Overview

Properties

<code>normal</code>	Normal of the plane.
<code>origin</code>	Origin of the plane.

Import detail

```
from ansys.tools.visualization_interface.utils.clip_plane import ClipPlane
```

Property detail

property ClipPlane.**normal**: Tuple[float, float, float]

Normal of the plane.

Returns

Tuple[float, float, float]

Normal of the plane.

property ClipPlane.origin: **Tuple**[float, float, float]

Origin of the plane.

Returns

Tuple[float, float, float]

Origin of the plane.

Description

Provides the ClipPlane class.

The color.py module

Summary

Enums

<i>Color</i>	Provides an enum with the color to use for the plotter actors.
--------------	--

Color

class ansys.tools.visualization_interface.utils.color.Color(*args, **kwargs)

Bases: `enum.Enum`

Provides an enum with the color to use for the plotter actors.

Overview

Attributes

<i>DEFAULT</i>	Default color for the plotter actors.
<i>PICKED</i>	Color for the actors that are currently picked.
<i>EDGE</i>	Default color for the edges.
<i>PICKED_EDGE</i>	Color for the edges that are currently picked.

Import detail

```
from ansys.tools.visualization_interface.utils.color import Color
```

Attribute detail

Color.DEFAULT = '#D6F7D1'

Default color for the plotter actors.

Color.PICKED = '#BB6EEE'

Color for the actors that are currently picked.

Color.EDGE = '#000000'

Default color for the edges.

`Color.PICKED_EDGE = '#9C9C9C'`

Color for the edges that are currently picked.

Description

Provides an enum with the color to use for the plotter actors.

The `logger.py` module

Summary

Classes

<i>SingletonType</i>	Provides the singleton helper class for the logger.
<i>VizLogger</i>	Provides the singleton logger for the visualizer.

Attributes

<i>logger</i>

SingletonType

class `ansys.tools.visualization_interface.utils.logger.SingletonType`

Bases: `type`

Provides the singleton helper class for the logger.

Overview

Special methods

<code>__call__</code>	Call to redirect new instances to the singleton instance.
-----------------------	---

Import detail

```
from ansys.tools.visualization_interface.utils.logger import SingletonType
```

Method detail

`SingletonType.__call__(*args, **kwargs)`

Call to redirect new instances to the singleton instance.

VizLogger

class `ansys.tools.visualization_interface.utils.logger.VizLogger`(*level*: `int` = `logging.ERROR`,
logger_name: `str` =
`'VizLogger'`)

Bases: `object`

Provides the singleton logger for the visualizer.

Parameters**to_file**

[bool, default: `False`] Whether to include the logs in a file.

Overview**Methods**

<code>get_logger</code>	Get the logger.
<code>set_level</code>	Set the logger output level.
<code>enable_output</code>	Enable logger output to a given stream.
<code>add_file_handler</code>	Save logs to a file in addition to printing them to the standard output.

Import detail

```
from ansys.tools.visualization_interface.utils.logger import VizLogger
```

Method detail

`VizLogger.get_logger()`

Get the logger.

Returns**Logger**

Logger.

`VizLogger.set_level(level: int)`

Set the logger output level.

Parameters**level**

[int] Output Level of the logger.

`VizLogger.enable_output(stream=None)`

Enable logger output to a given stream.

If a stream is not specified, `sys.stderr` is used.

Parameters

stream: TextIO, default: `sys.stderr`

Stream to output the log output to.

`VizLogger.add_file_handler(logs_dir: str = './.log')`

Save logs to a file in addition to printing them to the standard output.

Parameters**logs_dir**

[str, default: `"./.log"`] Directory of the logs.

Description

Provides the singleton helper class for the logger.

Module detail

`logger.logger`

Description

Provides the Utils package.

The `plotter.py` module

Summary

Classes

<code>Plotter</code>	Base plotting class containing common methods and attributes.
----------------------	---

Plotter

```
class ansys.tools.visualization_interface.plotter.Plotter(backend: an-  
sys.tools.visualization_interface.backends._base.BaseBackend  
= None)
```

Base plotting class containing common methods and attributes.

This class is responsible for plotting objects using the specified backend.

Parameters

backend

[BaseBackend, optional] Plotting backend to use, by default PyVistaBackend.

Overview

Methods

<code>plot</code>	Plots an object using the specified backend.
<code>show</code>	Show the plotted objects.

Properties

<code>backend</code>	Return the base plotter object.
----------------------	---------------------------------

Import detail

```
from ansys.tools.visualization_interface.plotter import Plotter
```

Property detail

property `Plotter.backend`

Return the base plotter object.

Method detail

`Plotter.plot(plottable_object: Any, **plotting_options)`

Plots an object using the specified backend.

Parameters

`plottable_object`

[Any] Object to plot.

`plotting_options`

[dict] Additional plotting options.

`Plotter.show(plottable_object: Any = None, screenshot: str = None, name_filter: bool = None, **plotting_options) → None`

Show the plotted objects.

Parameters

`plottable_object`

[Any, optional] Object to show, by default None.

`screenshot`

[str, optional] Path to save a screenshot, by default None.

`name_filter`

[bool, optional] Flag to filter the object, by default None.

`plotting_options`

[dict] Additional plotting options the selected backend accepts.

Description

Module for the Plotter class.

3.1.2 Description

Visualization Interface Tool is a Python client library for visualizing the results of Ansys simulations.

3.1.3 Module detail

`visualization_interface.USE_TRAME: bool = False`

`visualization_interface.DOCUMENTATION_BUILD: bool`

Whether the documentation is being built or not.

`visualization_interface.TESTING_MODE: bool`

Whether the library is being built or not, used to avoid showing plots while testing.

`visualization_interface.USE_HTML_BACKEND: bool`

Whether the library is being built or not, used to avoid showing plots while testing.

`visualization_interface.__version__`

EXAMPLES

This section show how to use the Visualization Interface Tool to perform many different types of operations.

BASIC USAGE EXAMPLES

These examples show how to use the general plotter included in the Visualization Interface Tool.

ADVANCED USAGE EXAMPLES

These examples show how to use the Visualization Interface Tool to postprocess simulation data.

6.1 Basic usage examples

These examples show how to use the general plotter included in the Visualization Interface Tool.

6.1.1 Use trame as a remote service

This example shows how to launch a trame service and use it as a remote service.

First, we need to launch the trame service. We can do this by running the following code:

```
# import required libraries
from ansys.tools.visualization_interface.backends.pyvista.trame_service import (
    TrameService,
)

# create a trame service, in whatever port is available in your system
ts = TrameService(websocket_port=8765)

# run the service
ts.run()
```

Now, we can send meshes and plotter to the trame service. We can do this by running the following code in a separate terminal:

```
# import required libraries
import time

import pyvista as pv

from ansys.tools.visualization_interface.backends.pyvista.trame_remote import (
    send_mesh,
    send_pl,
)

# create an example plotter
plotter = pv.Plotter()
plotter.add_mesh(pv.Cube())

# send some example meshes
```

(continues on next page)

(continued from previous page)

```
send_mesh(pv.Sphere())
send_mesh(pv.Sphere(center=(3, 0, 0)))
time.sleep(4)

# if we send a plotter, the previous meshes will be deleted.
send_pl(plotter)
```

Total running time of the script: (0 minutes 0.000 seconds)

6.1.2 Use a PyVista Qt backend

PyVista Qt is a package that extends the PyVista functionality through the usage of Qt. Qt applications operate in a separate thread than VTK, you can simultaneously have an active VTK plot and a non-blocking Python session.

This example shows how to use the PyVista Qt backend to create a plotter

```
import pyvista as pv

from ansys.tools.visualization_interface import Plotter
from ansys.tools.visualization_interface.backends.pyvista import PyVistaBackend
```

Open a pyvistaqt window

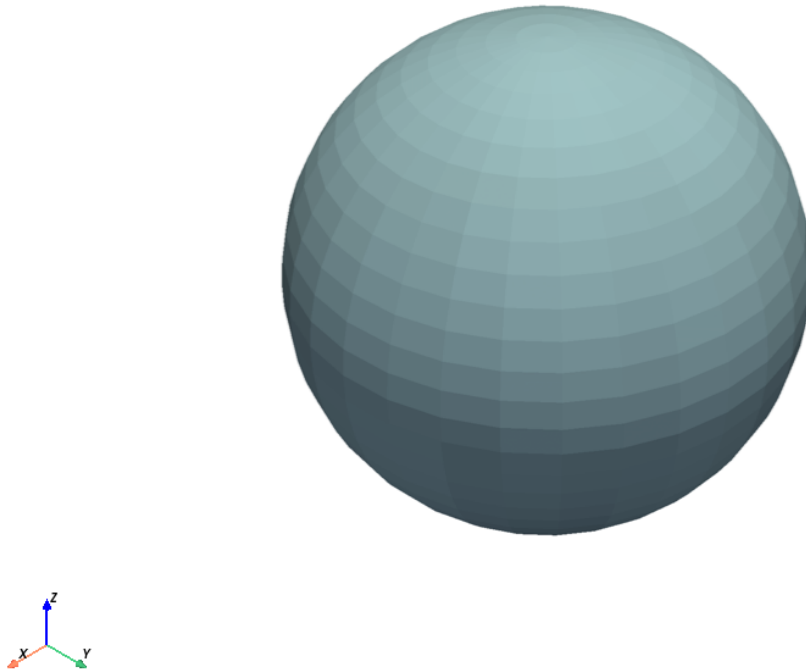
```
cube = pv.Cube()
pv_backend = PyVistaBackend(use_qt=True, show_qt=True)
pl = Plotter(backend=pv_backend)
pl.plot(cube)
pl.backend.enable_widgets()
pv_backend.scene.show()
```

Parallel VTK window

```
sphere = pv.Sphere()

pl_parallel = Plotter()
pl_parallel.plot(sphere)
pl_parallel.show()
```

Static Scene



Interactive Scene

Close the pyvistaqt window

```
pv_backend.close()
```

Integrate the plotter in a Qt application

```
pv_backend = PyVistaBackend(use_qt=True, show_qt=False)
pv_backend.enable_widgets()

# You can use this plotter in a Qt application
pl = pv_backend.scene
```

Total running time of the script: (0 minutes 6.148 seconds)

6.1.3 Use a clipping plane

This example shows how to use a clipping plane in the Visualization Interface Tool to cut a mesh.

```
import pyvista as pv

from ansys.tools.visualization_interface import ClipPlane, Plotter

mesh = pv.Cylinder()
```

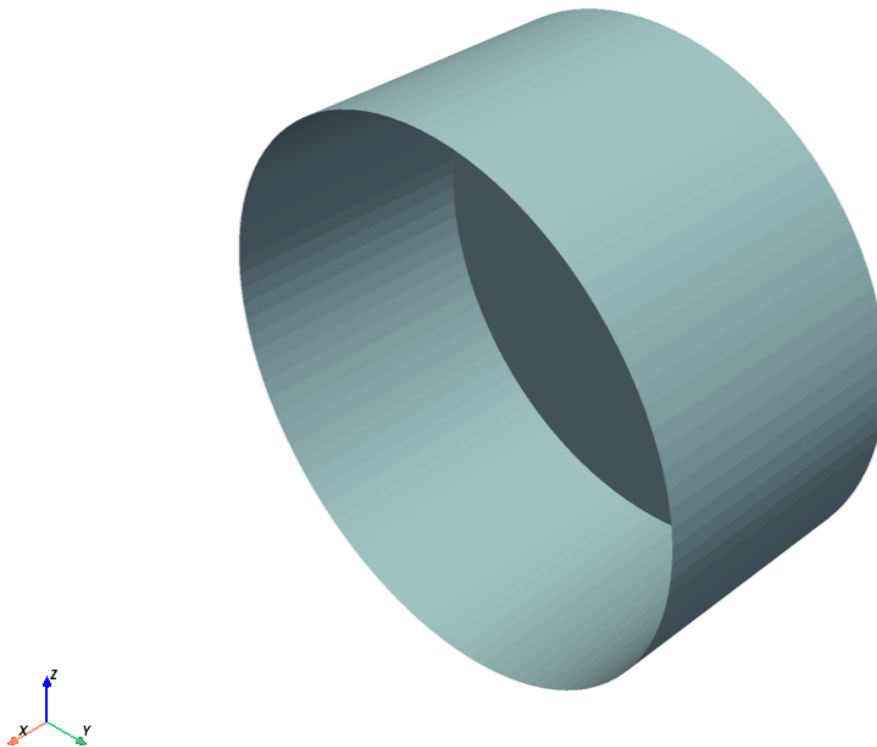
Create a plotter and clip the mesh

```
pl = Plotter()

# Create a clipping plane
clipping_plane = ClipPlane(normal=(1, 0, 0), origin=(0, 0, 0))

# Add the mesh to the plotter with the clipping plane
pl.plot(mesh, clipping_plane=clipping_plane)
pl.show()
```

Static Scene



Interactive Scene

Total running time of the script: (0 minutes 0.419 seconds)

6.1.4 Use the MeshObjectPlot class

The Visualization Interface Tool provides the MeshObject helper class to relate a custom object with its mesh. With a custom object, you can take advantage of the full potential of the Visualization Interface Tool.

This example shows how to use the MeshObjectPlot class to plot your custom objects.

Relate CustomObject class with a PyVista mesh

```
import pyvista as pv

# Note that the ``CustomObject`` class must have a way to get the mesh
# and a name or ID.

class CustomObject:
    def __init__(self):
        self.name = "CustomObject"
        self.mesh = pv.Cube()

    def get_mesh(self):
        return self.mesh

    def name(self):
        return self.name

# Create a custom object
custom_object = CustomObject()
```

Create a MeshObjectPlot instance

```
from ansys.tools.visualization_interface import MeshObjectPlot

# Create an instance

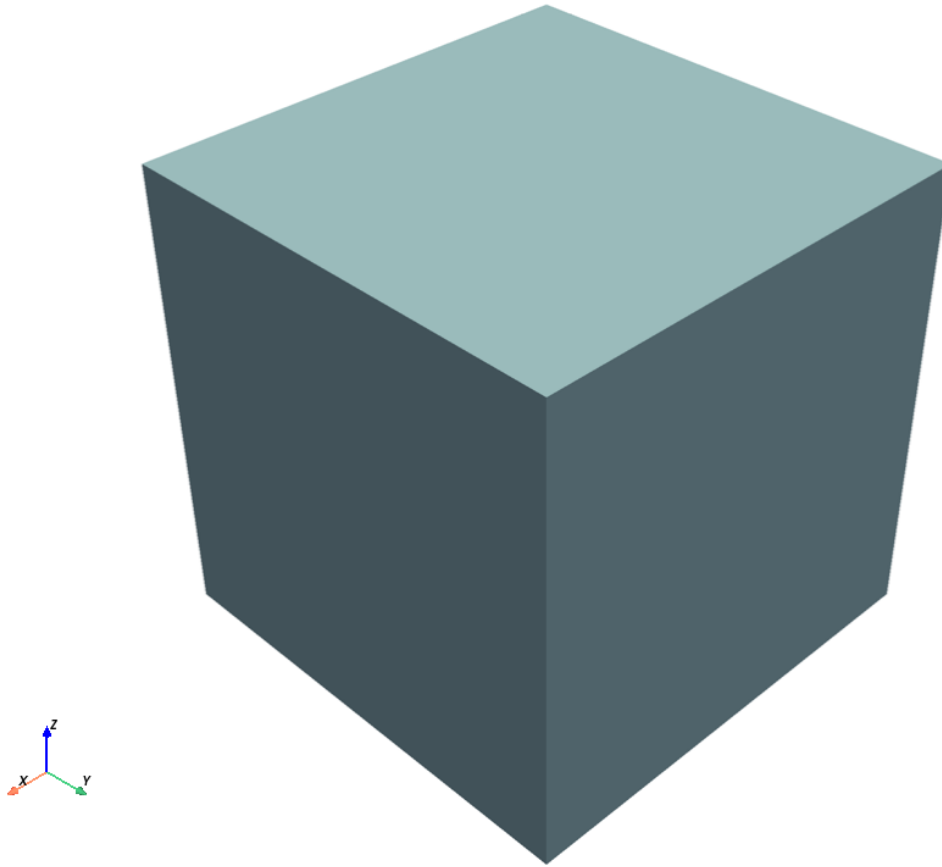
mesh_object = MeshObjectPlot(custom_object, custom_object.get_mesh())
```

Plot the MeshObjectPlot instance

```
from ansys.tools.visualization_interface import Plotter

pl = Plotter()
pl.plot(mesh_object)
pl.show()
```

Static Scene



Interactive Scene

Total running time of the script: (0 minutes 0.324 seconds)

6.1.5 Use the plotter

This example shows how to add one or more meshes to the plotter.

Add a mesh to the plotter

This code shows how to add a single mesh to the plotter.

```
import pyvista as pv

from ansys.tools.visualization_interface import Plotter

mesh = pv.Cube()

# Create a plotter
pl = Plotter()

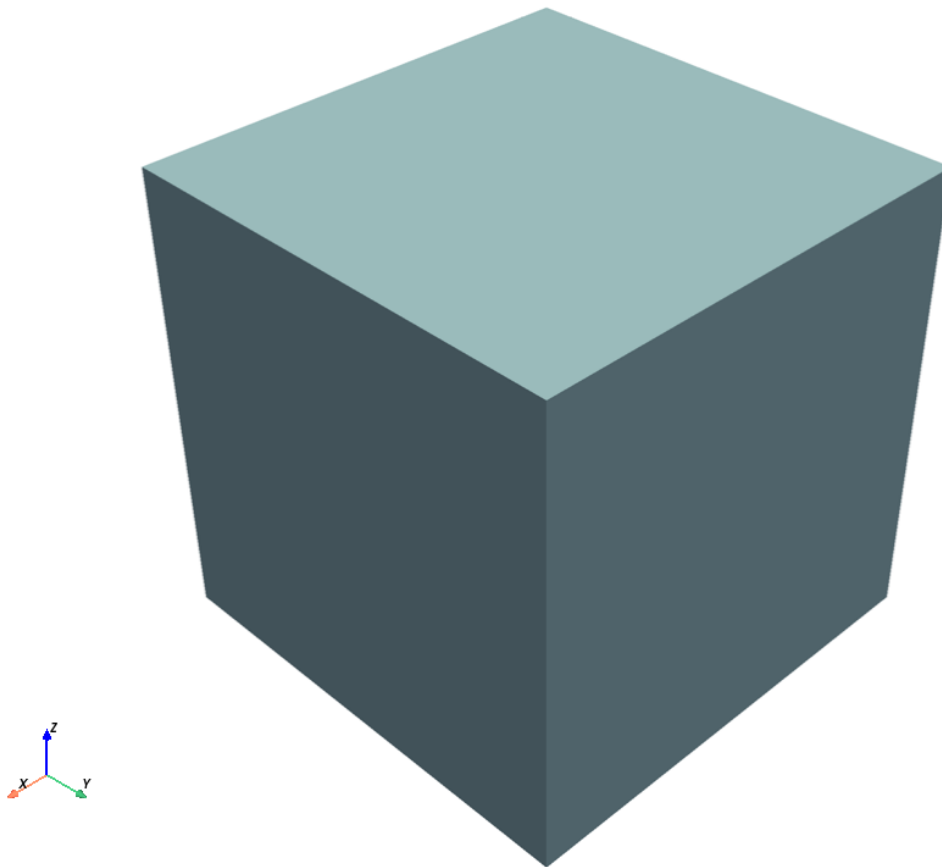
# Add the mesh to the plotter
pl.plot(mesh)
```

(continues on next page)

(continued from previous page)

```
# Show the plotter  
pl.show()
```

Static Scene



Interactive Scene

Getting a screenshot

Now we will check how to get a screenshot from our plotter.

```
import pyvista as pv  
  
from ansys.tools.visualization_interface import Plotter  
  
mesh = pv.Cube()  
  
# Create a plotter  
pl = Plotter()  
  
# Add the mesh to the plotter
```

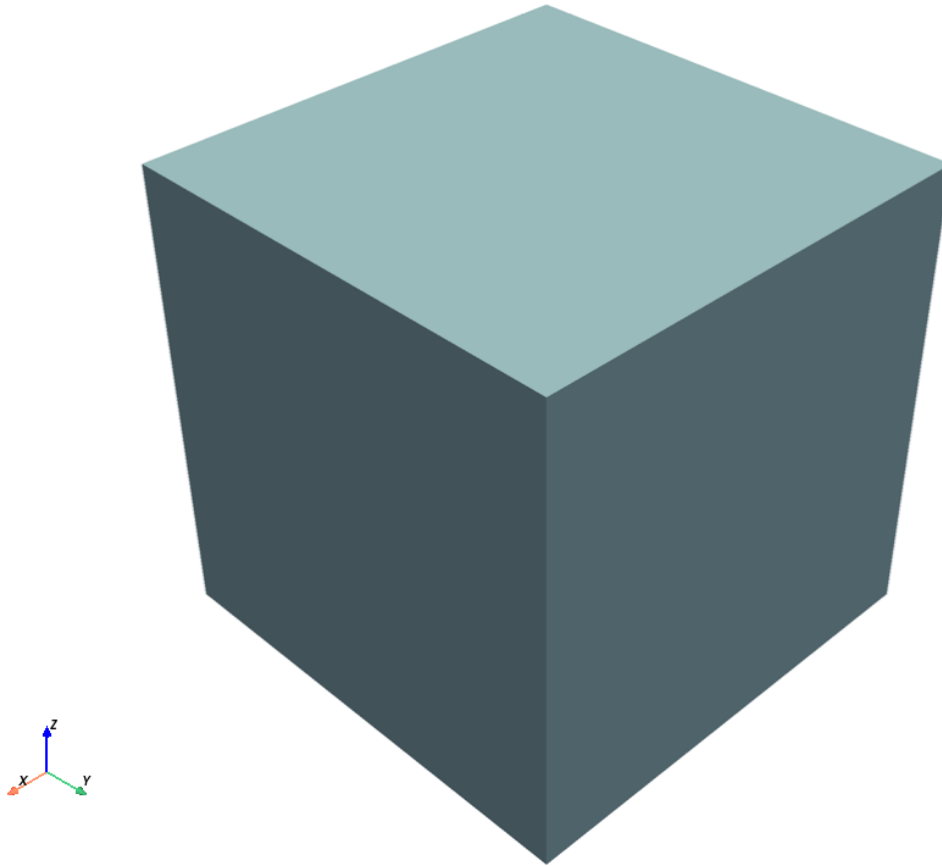
(continues on next page)

(continued from previous page)

```
pl.plot(mesh)

# Show the plotter
pl.show(screenshot="screenshot.png")
```

Static Scene



Interactive Scene

Add a list of meshes

This code shows how to add a list of meshes to the plotter.

```
import pyvista as pv

from ansys.tools.visualization_interface import Plotter

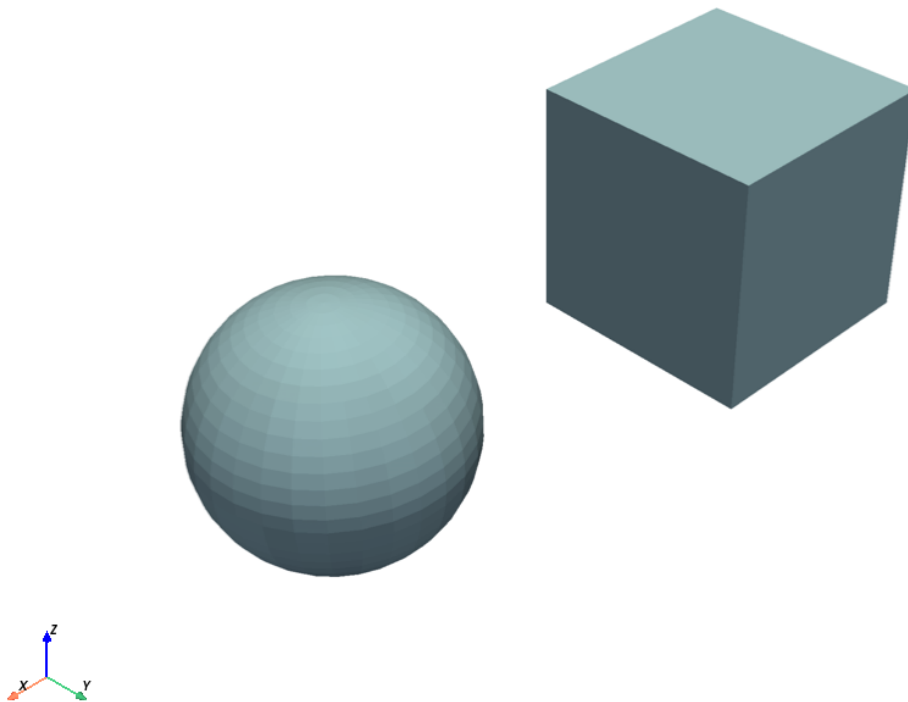
mesh1 = pv.Cube()
mesh2 = pv.Sphere(center=(2, 0, 0))
mesh_list = [mesh1, mesh2]
# Create a plotter
pl = Plotter()
```

(continues on next page)

(continued from previous page)

```
# Add a list of meshes to the plotter  
pl.plot(mesh_list)  
  
# Show the plotter  
pl.show()
```

Static Scene



Interactive Scene

Total running time of the script: (0 minutes 1.011 seconds)

6.1.6 Activate the picker

This example shows how to activate the picker, which is the tool that you use to select an object in the plotter and get its name.

Relate CustomObject class with a PyVista mesh

```
import pyvista as pv

# Note that the ``CustomObject`` class must have a way to get the mesh
# and a name or ID.

class CustomObject:
    def __init__(self):
        self.name = "CustomObject"
        self.mesh = pv.Cube(center=(1, 1, 0))

    def get_mesh(self):
        return self.mesh

    def name(self):
        return self.name

# Create a custom object
custom_cube = CustomObject()
custom_cube.name = "CustomCube"
custom_sphere = CustomObject()
custom_sphere.mesh = pv.Sphere(center=(0, 0, 5))
custom_sphere.name = "CustomSphere"
```

Create two MeshObjectPlot instances

```
from ansys.tools.visualization_interface import MeshObjectPlot

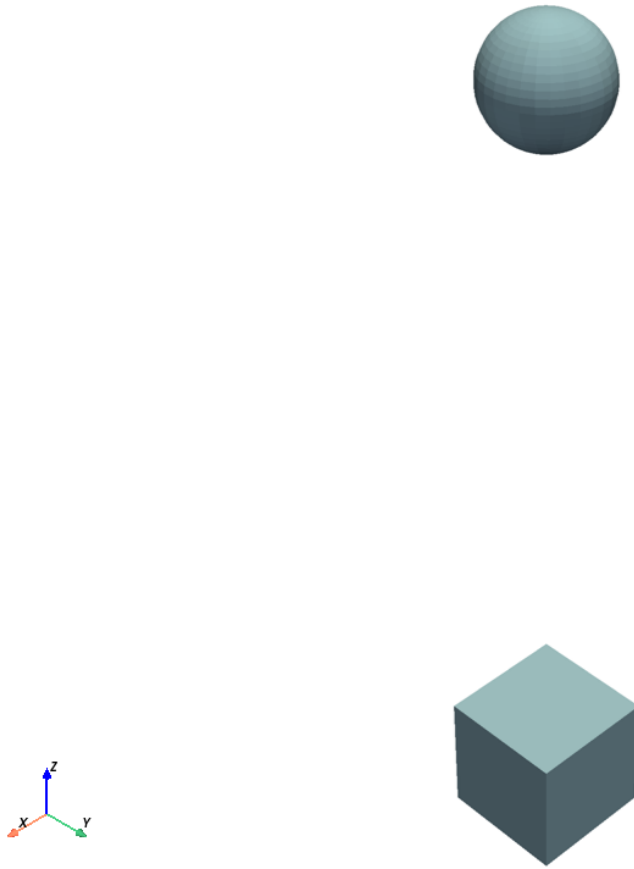
# Create an instance
mesh_object_cube = MeshObjectPlot(custom_cube, custom_cube.get_mesh())
mesh_object_sphere = MeshObjectPlot(custom_sphere, custom_sphere.get_mesh())
```

Activate the picking capabilities

```
from ansys.tools.visualization_interface import Plotter
from ansys.tools.visualization_interface.backends.pyvista import PyVistaBackend

pv_backend = PyVistaBackend(allow_picking=True, plot_picked_names=True)
pl = Plotter(backend=pv_backend)
pl.plot(mesh_object_cube)
pl.plot(mesh_object_sphere)
pl.show()
```

Static Scene



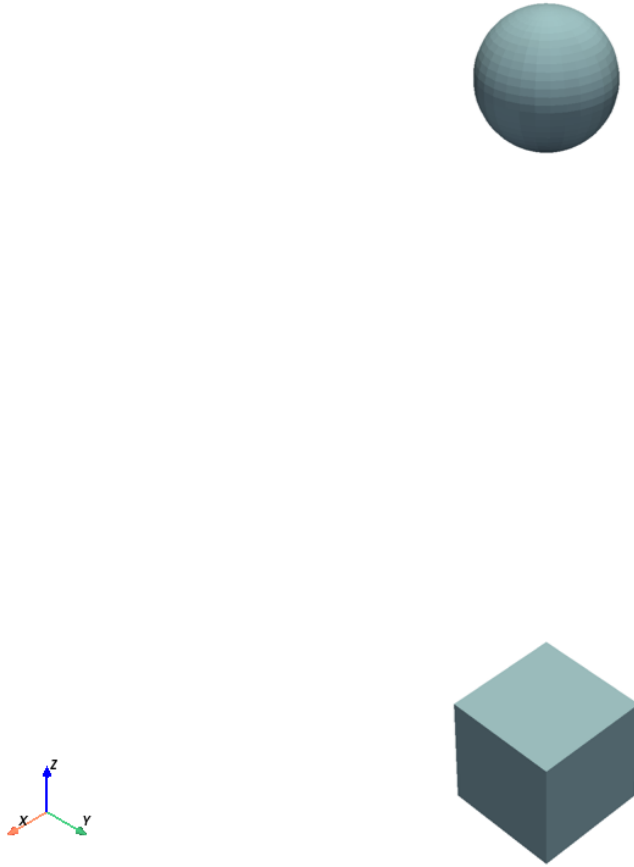
Interactive Scene

Activate the hover capabilities

```
from ansys.tools.visualization_interface import Plotter
from ansys.tools.visualization_interface.backends.pyvista import PyVistaBackend

pv_backend = PyVistaBackend(allow_hovering=True)
pl = Plotter(backend=pv_backend)
pl.plot(mesh_object_cube)
pl.plot(mesh_object_sphere)
pl.show()
```

Static Scene



Interactive Scene

Using StructuredGrid mesh

```
import numpy as np

class CustomStructuredObject:
    def __init__(self):
        self.name = "CustomObject"
        xrng = np.arange(-10, 10, 2, dtype=np.float32)
        yrng = np.arange(-10, 10, 5, dtype=np.float32)
        zrng = np.arange(-10, 10, 1, dtype=np.float32)
        x, y, z = np.meshgrid(xrng, yrng, zrng, indexing='ij')
        grid = pv.StructuredGrid(x, y, z)
        self.mesh = grid

    def get_mesh(self):
        return self.mesh

    def name(self):
```

(continues on next page)

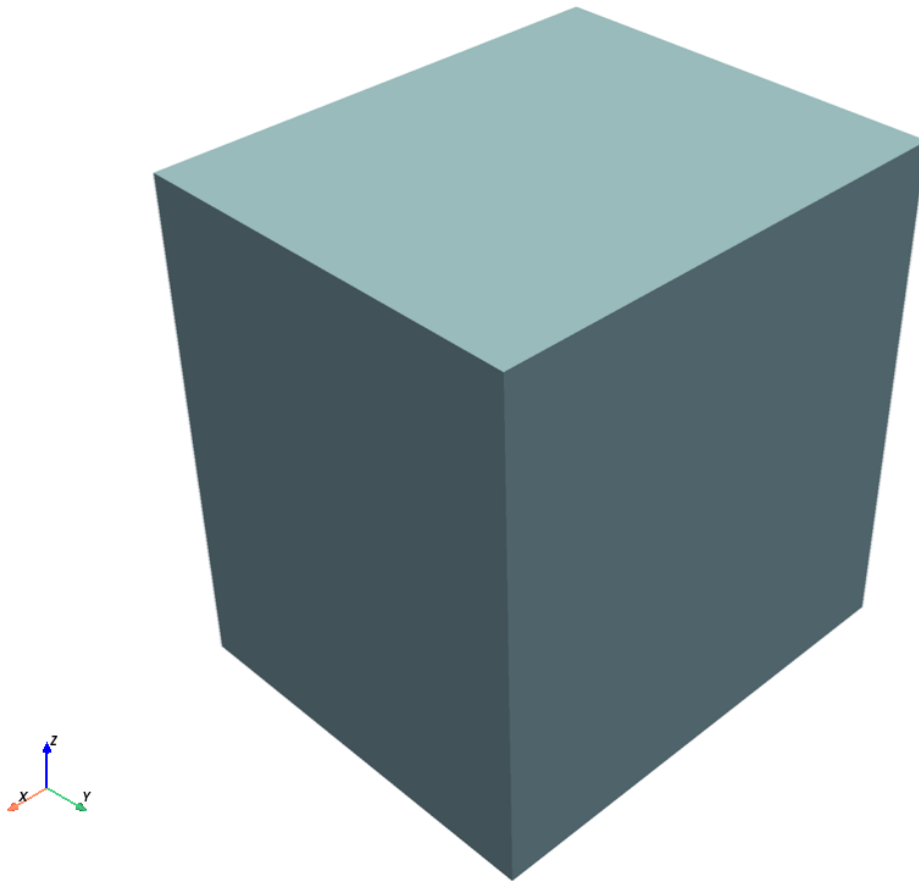
(continued from previous page)

```
return self.name

pv_backend = PyVistaBackend()
pl = Plotter(backend=pv_backend)

structured_object = CustomStructuredObject()
mo_plot = MeshObjectPlot(structured_object, structured_object.get_mesh())
pl.plot(mo_plot)
pl.show()
```

Static Scene



Interactive Scene

Total running time of the script: (0 minutes 0.885 seconds)

6.2 Advanced usage examples

These examples show how to use the Visualization Interface Tool to postprocess simulation data.

6.2.1 Postprocessing simulation results using the MeshObjectPlot class

The Visualization Interface Tool provides the `MeshObject` helper class to relate a custom object. With a custom object, you can take advantage of the full potential of the Visualization Interface Tool.

This example shows how to use the `MeshObjectPlot` class to plot your custom objects with scalar data on mesh.

Necessary imports

```
from ansys.fluent.core import examples
import pyvista as pv

from ansys.tools.visualization_interface.backends.pyvista import PyVistaBackend
from ansys.tools.visualization_interface import MeshObjectPlot, Plotter
```

Download the VTK file

A VTK dataset can be produced utilizing `PyDPF` for Ansys Flagship products simulations results file format.

```
mixing_elbow_file_src = examples.download_file("mixing_elbow.vtk", "result_files/fluent-
↪mixing_elbow_steady-state")
```

Define a custom object class

Note that the `CustomObject` class must have a way to get the mesh and a name or ID.

```
class CustomObject:
    def __init__(self):
        self.name = "CustomObject"
        self.mesh = pv.read(mixing_elbow_file_src)

    def get_mesh(self):
        return self.mesh

    def get_field_array_info(self):
        return self.mesh.array_names

    def name(self):
        return self.name

# Create a custom object
custom_vtk = CustomObject()
```

Create a MeshObjectPlot instance

```
mesh_object = MeshObjectPlot(custom_vtk, custom_vtk.get_mesh())

# Define the camera position
cpos = (
    (-0.3331763564757694, 0.08802797061044923, -1.055269197114142),
    (0.08813476356878325, -0.03975174212669032, -0.012819952697089087),
    (0.045604530283921085, 0.9935979348314435, 0.10336039239608838),
)
```

Get the available field data arrays

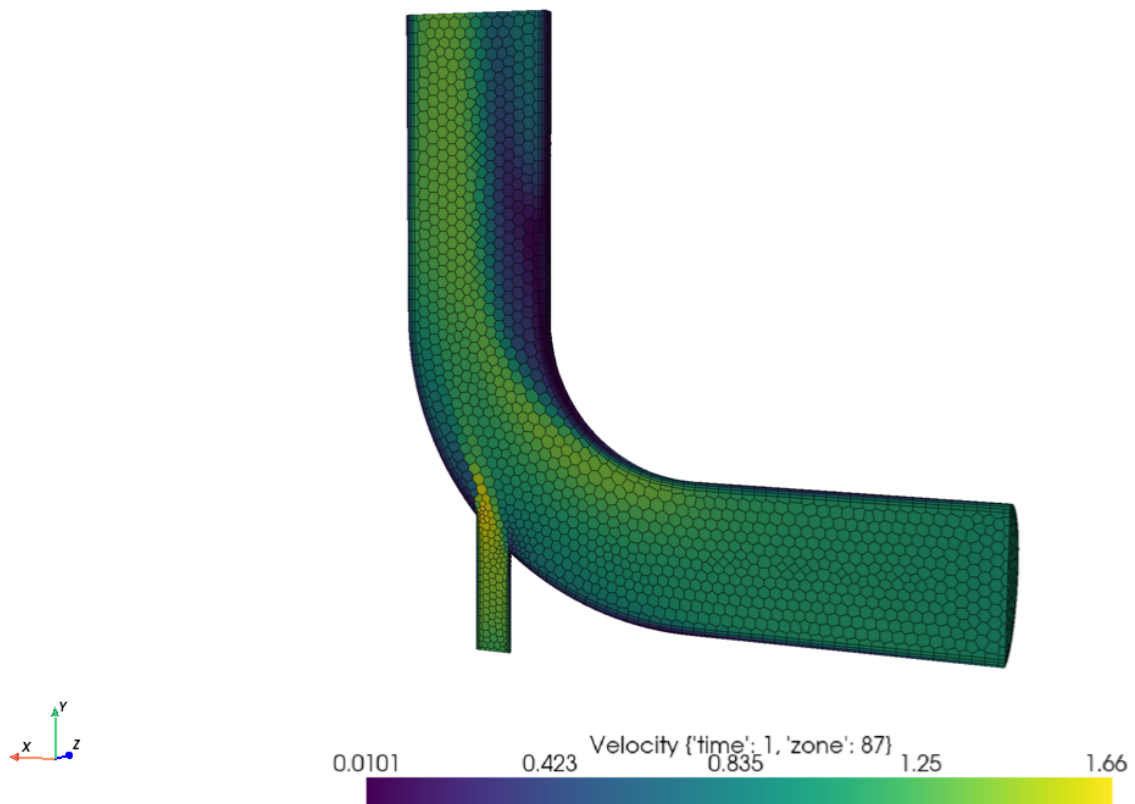
```
field_data_arrays = custom_vtk.get_field_array_info()
print(f"Field data arrays: {field_data_arrays}")
```

```
Field data arrays: ["Velocity {'time': 1, 'zone': 87}", "Temperature {'time': 1, 'zone': 87}"]
```

Plot the MeshObjectPlot instance with mesh object & field data (0)

```
pv_backend = PyVistaBackend()
pl = Plotter(backend=pv_backend)
pl.plot(
    mesh_object,
    scalars=field_data_arrays[0],
    show_edges=True,
    show_scalar_bar=True,
)
pl._backend.pv_interface.scene.camera_position = cpos
pl.show()
```

Static Scene

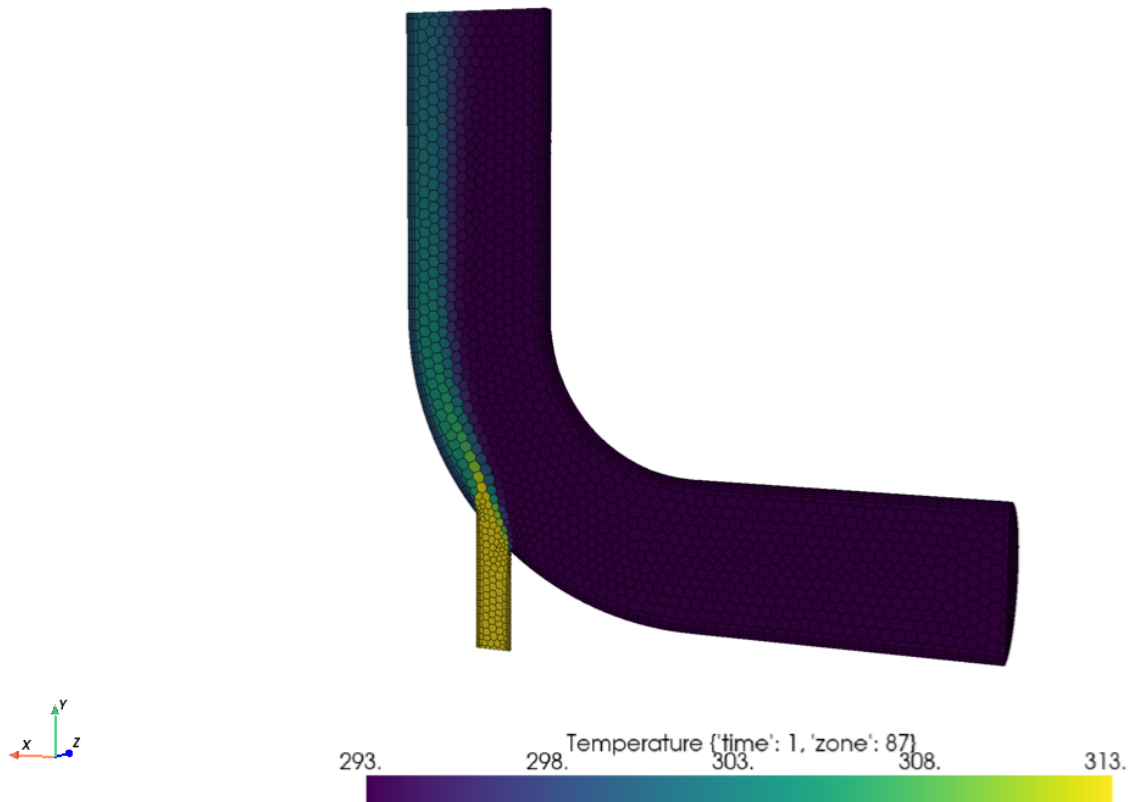


Interactive Scene

Plot the MeshObjectPlot instance with mesh object & other field data (1)

```
pv_backend = PyVistaBackend()
pl = Plotter(backend=pv_backend)
pl.plot(
    mesh_object,
    scalars=field_data_arrays[1],
    show_edges=True,
    show_scalar_bar=True,
)
pl._backend.pv_interface.scene.camera_position = cpos
pl.show()
```

Static Scene



Interactive Scene

Total running time of the script: (0 minutes 6.758 seconds)

CONTRIBUTE

Overall guidance on contributing to a PyAnsys library appears in the [Contributing](#) topic in the *PyAnsys developer's guide*. Ensure that you are thoroughly familiar with this guide before attempting to contribute to the Visualization Interface Tool.

The following contribution information is specific to the Visualization Interface Tool.

7.1 Install in developer mode

Installing the Visualization Interface Tool in developer mode allows you to modify and enhance the source.

To clone and install the latest Visualization Interface Tool release in development mode, run these commands:

```
git clone https://github.com/ansys/ansys-tools-visualization-interface
cd ansys-tools-visualization-interface
python -m pip install --upgrade pip
pip install -e .
```

7.2 Run tests

The Visualization Interface Tool uses [pytest](#) for testing.

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

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

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

```
pytest
```

7.3 Adhere to code style

The Visualization Interface Tool 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:

```
$ pre-commit install
$ git commit -am "added my cool feature"
black.....Passed
blacken-docs.....Passed
isort.....Passed
flake8.....Passed
docformatter.....Passed
codespell.....Passed
pydocstyle.....Passed
check for merge conflicts.....Passed
debug statements (python).....Passed
check yaml.....Passed
trim trailing whitespace.....Passed
Add License Headers.....Passed
Validate GitHub Workflows.....Passed
```

7.4 Build the documentation

You can build the Visualization Interface Tool documentation locally.

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

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

2. To then build the documentation, 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
```

7.5 Post issues

Use the [Visualization Interface Tool Issues](#) page to report bugs and request new features. When possible, use the issue templates provided. If your issue does not fit into one of these templates, click the link for opening a blank issue.

If you have general questions about the PyAnsys ecosystem, email pyansys.core@ansys.com. If your question is specific to the Visualization Interface Tool, ask your question in an issue as described in the previous paragraph.

PYTHON MODULE INDEX

a

ansys.tools.visualization_interface, 7
ansys.tools.visualization_interface.backends, 7
ansys.tools.visualization_interface.backends.pyvista, 8
ansys.tools.visualization_interface.backends.pyvista.pyvista, 21
ansys.tools.visualization_interface.backends.pyvista.pyvista_interface, 30
ansys.tools.visualization_interface.backends.pyvista.trame_local, 35
ansys.tools.visualization_interface.backends.pyvista.trame_remote, 36
ansys.tools.visualization_interface.backends.pyvista.trame_service, 37
ansys.tools.visualization_interface.backends.pyvista.widgets, 8
ansys.tools.visualization_interface.backends.pyvista.widgets.button, 8
ansys.tools.visualization_interface.backends.pyvista.widgets.displace_arrows, 10
ansys.tools.visualization_interface.backends.pyvista.widgets.hide_buttons, 12
ansys.tools.visualization_interface.backends.pyvista.widgets.measure, 13
ansys.tools.visualization_interface.backends.pyvista.widgets.mesh_slider, 14
ansys.tools.visualization_interface.backends.pyvista.widgets.ruler, 16
ansys.tools.visualization_interface.backends.pyvista.widgets.screenshot, 17
ansys.tools.visualization_interface.backends.pyvista.widgets.view_button, 18
ansys.tools.visualization_interface.backends.pyvista.widgets.widget, 20
ansys.tools.visualization_interface.plotter, 46
ansys.tools.visualization_interface.types, 38
ansys.tools.visualization_interface.types.edge_plot, 39
ansys.tools.visualization_interface.types.mesh_object_plot, 40
ansys.tools.visualization_interface.utils, 42
ansys.tools.visualization_interface.utils.clip_plane, 42
ansys.tools.visualization_interface.utils.color, 43
ansys.tools.visualization_interface.utils.logger, 44

Symbols

`__call__()` (in module *SingletonType*), 44

`__version__` (in module *visualization_interface*), 47

A

`actor` (in module *EdgePlot*), 39

`actor` (in module *MeshObjectPlot*), 41

`add_file_handler()` (in module *VizLogger*), 45

`add_widget()` (in module *PyVistaBackendInterface*), 25

`ansys.tools.visualization_interface`

module, 7

`ansys.tools.visualization_interface.backends`

module, 7

`ansys.tools.visualization_interface.backends.pyvista`

module, 8

`ansys.tools.visualization_interface.backends.pyvista.pyvista`

module, 21

`ansys.tools.visualization_interface.backends.pyvista.pyvista.PyVistaBackend`

(built-in class), 28

`ansys.tools.visualization_interface.backends.pyvista.pyvista.PyVistaBackendInterface`

(built-in class), 22

`ansys.tools.visualization_interface.backends.pyvista.pyvista_interface`

module, 30

`ansys.tools.visualization_interface.backends.pyvista.pyvista_interface.PyVistaInterface`

(built-in class), 30

`ansys.tools.visualization_interface.backends.pyvista.pyvista.frame_local`

module, 35

`ansys.tools.visualization_interface.backends.pyvista.pyvista.frame_local.FrameVisualizer`

(built-in class), 35

`ansys.tools.visualization_interface.backends.pyvista.pyvista.frame_remote`

module, 36

`ansys.tools.visualization_interface.backends.pyvista.pyvista.frame_service`

module, 37

`ansys.tools.visualization_interface.backends.pyvista.pyvista.frame_service.FrameService`

(built-in class), 37

`ansys.tools.visualization_interface.backends.pyvista.pyvista.widgets`

module, 8

`ansys.tools.visualization_interface.backends.pyvista.pyvista.widgets.button`

module, 8

`ansys.tools.visualization_interface.backends.pyvista.pyvista.widgets.button.Button`

(built-in class), 8

`ansys.tools.visualization_interface.backends.pyvista.pyvista.widgets.displace_arrows`

module, 10

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 11

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 10

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 12

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 12

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 13

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 13

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 14

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 15

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 16

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 16

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 17

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 17

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 18

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 18

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 19

`ansys.tools.visualization_interface.backends.pyvista.widgets`

module, 20

`ansys.tools.visualization_interface.backends.pyvista.widgets`

(built-in class), 20

`ansys.tools.visualization_interface.plotter`

module, 46

`ansys.tools.visualization_interface.plotter.Plotter`

(built-in class), 46

`ansys.tools.visualization_interface.types`

module, 38

`ansys.tools.visualization_interface.types.edge_plot`

module, 39
 ansys.tools.visualization_interface.types.edge_plot.EdgePlot
 (built-in class), 39
 ansys.tools.visualization_interface.types.mesh_object_plot.
 module, 40
 ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot
 (built-in class), 40
 ansys.tools.visualization_interface.utils
 module, 42
 ansys.tools.visualization_interface.utils.clip_plane
 module, 42
 ansys.tools.visualization_interface.utils.clip_plane.ClipPlane
 (built-in class), 42
 ansys.tools.visualization_interface.utils.color
 module, 43
 ansys.tools.visualization_interface.utils.color.Color
 (built-in class), 43
 ansys.tools.visualization_interface.utils.logger
 module, 44
 ansys.tools.visualization_interface.utils.logger.SingletonType
 (built-in class), 44
 ansys.tools.visualization_interface.utils.logger.VizLogger
 (built-in class), 44

B
 backend (in module Plotter), 47
 base_plotter (in module PyVistaBackend), 29
 button_config (in module Button), 9

C
 callback() (in module Button), 9
 callback() (in module DisplacementArrow), 11
 callback() (in module HideButton), 13
 callback() (in module MeasureWidget), 14
 callback() (in module MeshSliderWidget), 15
 callback() (in module PlotterWidget), 21
 callback() (in module Ruler), 16
 callback() (in module ScreenshotButton), 18
 callback() (in module ViewButton), 19
 clear_plotter() (in module TrameService), 38
 CLIENT_TYPE (in module trame_local), 36
 clip() (in module PyVistaInterface), 33
 close() (in module PyVistaBackend), 29
 compute_edge_object_map() (in module
 PyVistaBackendInterface), 26
 custom_object (in module MeshObjectPlot), 41

D
 DARK_MODE_THRESHOLD (in module pyvista), 30
 DEFAULT (in module Color), 43
 direction (in module DisplacementArrow), 11
 direction (in module ViewButton), 19
 disable_hover() (in module PyVistaBackendInter-
 face), 26
 disable_picking() (in module PyVistaBackendInter-
 face), 26
 DOCUMENTATION_BUILD (in module visualiza-
 tion_interface), 47

E
 EdgePlot (in module Color), 43
 edge_object (in module EdgePlot), 39
 edges (in module MeshObjectPlot), 41
 enable_hover() (in module PyVistaBackendInterface),
 26
 enable_output() (in module VizLogger), 45
 enable_picking() (in module PyVistaBackendInter-
 face), 26
 enable_widgets() (in module PyVistaBackendInter-
 face), 25

G
 get_logger() (in module VizLogger), 45

H
 hover_callback() (in module PyVistaBackendInter-
 face), 26

I
 ISOMETRIC (in module ViewDirection), 20

L
 logger (in module logger), 46

M
 mesh (in module MeshObjectPlot), 41
 module
 ansys.tools.visualization_interface, 7
 ansys.tools.visualization_interface.backends,
 7
 ansys.tools.visualization_interface.backends.pyvista,
 8
 ansys.tools.visualization_interface.backends.pyvista.p
 21
 ansys.tools.visualization_interface.backends.pyvista.p
 30
 ansys.tools.visualization_interface.backends.pyvista.t
 35
 ansys.tools.visualization_interface.backends.pyvista.t
 36
 ansys.tools.visualization_interface.backends.pyvista.t
 37
 ansys.tools.visualization_interface.backends.pyvista.w
 8
 ansys.tools.visualization_interface.backends.pyvista.w
 8

ansys.tools.visualization_interface.backends.pyvista.widgets.PyVistaInterface, 29
 10 plot_edges() (in module PyVistaInterface), 33
 ansys.tools.visualization_interface.backends.pyvista.widgets.HideButton, 29
 12 plot_iter() (in module PyVistaBackendInterface), 27
 ansys.tools.visualization_interface.backends.pyvista.widgets.MeasureWidget, 34
 13 plot_meshobject() (in module PyVistaInterface), 33
 ansys.tools.visualization_interface.backends.pyvista.widgets.MeshSlider, 14
 14 plotter_helper (in module MeasureWidget), 14
 ansys.tools.visualization_interface.backends.pyvista.widgets.MeshSliderWidget, 15
 16 pv_interface (in module PyVistaBackendInterface), 25
 ansys.tools.visualization_interface.backends.pyvista.widgets.screenshot, 17
 17 **R**
 ansys.tools.visualization_interface.backends.pyvista.widgets.view_button, 18
 18 **S**
 ansys.tools.visualization_interface.backends.pyvista.widgets.widget, 20
 20 scene (in module PyVistaBackendInterface), 25
 ansys.tools.visualization_interface.plotter, 46
 46 scene (in module PyVistaInterface), 32
 ansys.tools.visualization_interface.types, 38
 38 select_object() (in module PyVistaBackendInterface), 25
 ansys.tools.visualization_interface.types, 39
 39 send_mesh() (in module trame_remote), 37
 ansys.tools.visualization_interface.types, 39
 39 send_pl() (in module trame_remote), 36
 ansys.tools.visualization_interface.types, 40
 40 server (in module TrameVisualizer), 36
 ansys.tools.visualization_interface.types, 40
 40 set_add_mesh_defaults() (in module PyVistaInterface), 35
 ansys.tools.visualization_interface.utils, 42
 42 set_level() (in module VizLogger), 45
 ansys.tools.visualization_interface.utils, 42
 42 set_scene() (in module TrameService), 38
 ansys.tools.visualization_interface.utils, 42
 42 clip_plane, 36
 ansys.tools.visualization_interface.utils, 43
 43 show() (in module Plotter), 47
 ansys.tools.visualization_interface.utils, 43
 43 show() (in module PyVistaBackendInterface), 26
 ansys.tools.visualization_interface.utils, 44
 44 show() (in module PyVistaInterface), 34
 ansys.tools.visualization_interface.utils, 44
 44 show() (in module TrameVisualizer), 36
 show_plotter() (in module PyVistaBackendInterface), 27

N

name (in module EdgePlot), 40
 name (in module MeshObjectPlot), 41
 normal (in module ClipPlane), 42

O

object_to_actors_map (in module PyVistaInterface), 32
 origin (in module ClipPlane), 43

P

parent (in module EdgePlot), 40
 PICKED (in module Color), 43
 PICKED_EDGE (in module Color), 43
 picked_operation() (in module PyVistaBackendInterface), 28
 picker_callback() (in module PyVistaBackendInterface), 26
 plot() (in module Plotter), 47
 plot() (in module PyVistaBackend), 29
 plot() (in module PyVistaBackendInterface), 27

T

TESTING_MODE (in module visualization_interface), 47

U

unselect_object() (in module PyVistaBackendInterface), 25
 update() (in module Button), 9
 update() (in module HideButton), 13
 update() (in module MeasureWidget), 14
 update() (in module MeshSliderWidget), 15
 update() (in module PlotterWidget), 21
 update() (in module Ruler), 17
 update() (in module ScreenshotButton), 18
 USE_HTML_BACKEND (in module visualization_interface), 47
 USE_TRAME (in module visualization_interface), 47

V

view_xy() (in module PyVistaInterface), 33

[view_xz\(\)](#) (in module *PyVistaInterface*), 33
[view_yx\(\)](#) (in module *PyVistaInterface*), 33
[view_yz\(\)](#) (in module *PyVistaInterface*), 33
[view_zx\(\)](#) (in module *PyVistaInterface*), 33
[view_zy\(\)](#) (in module *PyVistaInterface*), 33

X

[XDOWN](#) (in module *CameraPanDirection*), 12
[XUP](#) (in module *CameraPanDirection*), 12
[XYMINUS](#) (in module *ViewDirection*), 20
[XYPLUS](#) (in module *ViewDirection*), 20
[XZMINUS](#) (in module *ViewDirection*), 20
[XZPLUS](#) (in module *ViewDirection*), 20

Y

[YDOWN](#) (in module *CameraPanDirection*), 12
[YUP](#) (in module *CameraPanDirection*), 12
[YZMINUS](#) (in module *ViewDirection*), 20
[YZPLUS](#) (in module *ViewDirection*), 20

Z

[ZDOWN](#) (in module *CameraPanDirection*), 12
[ZUP](#) (in module *CameraPanDirection*), 12