



© 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

Oct 30, 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 |
| 2.3 | Customizing the picker and hover callbacks | 7 |
| 3 | Migration | 9 |
| 3.1 | Code migration | 9 |
| 3.1.1 | Replace PyVista plotter code with Ansys Tools Visualization Interface plotter code | 9 |
| 3.1.2 | Convert your custom meshes to objects usable by the Ansys Tools Visualization Interface plotter | 10 |
| 3.1.3 | Customize the PyVista backend | 10 |
| 3.1.4 | Customize the picker or hover behavior | 12 |
| 3.1.5 | Use the PyVista Qt backend | 12 |
| 3.2 | Documentation configuration migration | 12 |
| 4 | API reference | 15 |
| 4.1 | The <code>ansys.tools.visualization_interface</code> library | 15 |
| 4.1.1 | Summary | 15 |
| 4.1.2 | Description | 72 |
| 4.1.3 | Module detail | 72 |
| 5 | Examples | 73 |
| 6 | Basic usage examples | 75 |
| 7 | Basic Plotly usage examples | 77 |
| 8 | Advanced usage examples | 79 |
| 8.1 | Basic usage examples | 79 |
| 8.1.1 | Use <code>trame</code> as a remote service | 79 |
| 8.1.2 | Use a PyVista Qt backend | 80 |
| 8.1.3 | Use a clipping plane | 82 |
| 8.1.4 | Use the <code>MeshObjectPlot</code> class | 83 |
| 8.1.5 | Use the plotter | 84 |
| 8.1.6 | <code>MeshObjectPlot</code> tree structure | 88 |
| 8.1.7 | Activate the picker | 89 |

| | | |
|----------|---|------------|
| 8.1.8 | Create custom picker | 94 |
| 8.2 | Basic Plotly usage examples | 98 |
| 8.2.1 | Plain usage of the plotly dash backend | 98 |
| 8.2.2 | Plain usage of the plotly backend | 99 |
| 8.3 | Advanced usage examples | 100 |
| 8.3.1 | Postprocessing simulation results using the <code>MeshObjectPlot</code> class | 100 |
| 9 | Contribute | 105 |
| 9.1 | Install in developer mode | 105 |
| 9.2 | Run tests | 105 |
| 9.3 | Adhere to code style | 105 |
| 9.4 | Build the documentation | 106 |
| 9.5 | Post issues | 106 |
| | Python Module Index | 107 |
| | Index | 109 |

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. For information on how to migrate from PyVista to the Ansys Visualization Interface Tool, see [Migration](#).

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 ##

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`.

For comprehensive migration information with code examples, see [Migration](#).

2.3 Customizing the picker and hover callbacks

The Visualization Interface Tool provides a base class, [AbstractPicker](#), for customizing the picker and hover callbacks of the plotter. This class provides a set of methods that can be overridden so that you can adapt the picker and hover functionalities to the specific need of your PyAnsys library.

The first thing you must do is to create a class that inherits from the [AbstractPicker](#) class. After that, see these main use cases for customizing the picker and hover callbacks:

- You may want to change the way that objects are picked in the plotter. To do this, you can override the `pick_select_object` and `pick_unselect_object` methods. These methods are called when an object is selected or unselected, respectively.
- Similarly, you may want to change the way that objects are hovered over in the plotter. To do this, you can override the `hover_select_object` and `hover_unselect_object` methods. These methods are called when an object is hovered over or unhovered, respectively.

A practical example of how to use the [AbstractPicker](#) class is included in [Create custom picker](#).

MIGRATION

This section helps you migrate from PyVista plotters to the Ansys Tools Visualization Interface plotters. It consists of two major topics:

- *Code migration*
- *Documentation configuration migration*

3.1 Code migration

This topic explains how to migrate from PyVista plotters to the new Ansys Tools Visualization Interface plotters. Because cases vary greatly, it provides a few examples that cover the most common scenarios.

3.1.1 Replace PyVista plotter code with Ansys Tools Visualization Interface plotter code

If you only need to plot simple PyVista meshes, you can directly replace your PyVista plotter code with the Ansys Tools Visualization Interface plotter code. On top of common PyVista functionalities, the Ansys Tools Visualization Interface plotter provides additional interactivity such as view buttons and mesh slicing.

The following code shows how to do the plotter code replacement:

- PyVista code:

```
import pyvista as pv

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

# Create a plotter
pl = pv.Plotter()

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

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

- Ansys Tools Visualization Interface code:

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

(continues on next page)

(continued from previous page)

```
# 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()
```

3.1.2 Convert your custom meshes to objects usable by the Ansys Tools Visualization Interface plotter

Your custom object must have a method that returns a PyVista mesh and a method that exposes a name or id attribute of your object:

```
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
```

You then need to create a *MeshObjectPlot* instance that relates the PyVista mesh with your custom object:

```
from ansys.tools.visualization_interface import MeshObjectPlot

custom_object = CustomObject()
mesh_object_plot = MeshObjectPlot(
    custom_object=custom_object,
    mesh=custom_object.get_mesh(),
)
```

With this, you can use the Ansys Tools Visualization Interface plotter to visualize your custom object. It enables interactivity such as picking and hovering.

3.1.3 Customize the PyVista backend

You can customize the backend of the Ansys Tools Visualization Interface plotter to enable or turn off certain functionalities. The following code shows how to enable picking:

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

backend = PyVistaBackend(allow_picking=True)
```

(continues on next page)

(continued from previous page)

```
# Create a plotter
pl = Plotter(backend=backend)

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

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

If you want to customize the backend even more, you can create your own backend by inheriting from the `PyVistaBackendInterface` class and implementing the required methods:

```
@abstractmethod
def plot_iter(self, 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 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
        :meth:`Plotter.add_mesh <pyvista.Plotter.add_mesh>` method.

    """
    pass

@abstractmethod
def plot(self, plottable_object: Any, name_filter: str = None, **plotting_options):
    """Plot a single object to the plotter.

    Parameters
    -----
    plottable_object : Any
        Object to plot.
    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
        :meth:`Plotter.add_mesh <pyvista.Plotter.add_mesh>` method.

    """
    pass
```

The rest of the methods are implemented for you. This ensures that while you can customize what you need for plotting, the rest of the functionalities still work as expected. For more information, see the backend documentation. If you need to even go further, you can create your own plotter by inheriting from the `BaseBackend` class and implementing the required methods, although this may break existing features.

3.1.4 Customize the picker or hover behavior

You can customize the picker of the Ansys Tools Visualization Interface plotter to decide what happens when you pick or hover over an object. For example, if you want to print the name of the picked object, you can do it as described in the *Create custom picker* example.

3.1.5 Use the PyVista Qt backend

You can use the PyVista Qt backend with the Ansys Tools Visualization Interface plotter. To do this, you must set the PyVista backend to Qt before creating the plotter:

```
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()
```

You can then integrate the plotter into a PyQt or PySide app by disabling the `show_qt` parameter. For more information about this, see the [PyVista documentation](#).

3.2 Documentation configuration migration

This topic explains how to migrate from the PyVista documentation configuration to the new Ansys Tools Visualization Interface documentation configuration.

1. Add environment variables for documentation:

```
os.environ["PYANSYS_VISUALIZER_DOC_MODE"] = "true"
os.environ["PYANSYS_VISUALIZER_HTML_BACKEND"] = "true"
```

2. Use PyVista DynamicScraper:

```
from pyvista.plotting.utilities.sphinx_gallery import DynamicScraper

sphinx_gallery_conf = {
    "image_scrapers": (DynamicScraper()),
}
```

3. Add PyVista viewer directive to extensions:

```
extensions = ["pyvista.ext.viewer_directive"]
```

4. Make sure you are executing the notebook cells:

```
nbsphinx_execute = "always"
```

For Plotly, in `conf.py`, do the following:

1. Add environment variables for documentation:

```
os.environ["PYANSYS_VISUALIZER_DOC_MODE"] = "true"
```

2. Add plotly configuration

```
import plotly.io as pio  
  
pio.renderers.default = "sphinx_gallery"
```

3. Import and add scraper

```
from plotly.io._sg_scraper import plotly_sg_scraper  
  
sphinx_gallery_conf = {  
    "image_scrapers": (DynamicScraper(), "matplotlib", plotly_sg_scraper),  
}
```

4. **[IMPORTANT]** The `pl.show()` must be the last line of code in the cell, or else it won't show.

API REFERENCE

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

4.1 The `ansys.tools.visualization_interface` library

4.1.1 Summary

Subpackages

| | |
|-----------------------|-----------------------------|
| <code>backends</code> | Provides interfaces. |
| <code>types</code> | Provides custom types. |
| <code>utils</code> | Provides the Utils package. |

Submodules

| | |
|----------------------|-------------------------------|
| <code>plotter</code> | Module for the Plotter class. |
|----------------------|-------------------------------|

Attributes

| |
|--------------------------|
| <code>__version__</code> |
|--------------------------|

Constants

| | |
|----------------------------------|---|
| <code>USE_TRAME</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

| | |
|----------------------|------------------------|
| <code>plotly</code> | Plotly initialization. |
| <code>pyvista</code> | Provides interfaces. |

The plotly package

Summary

Subpackages

| | |
|----------------|----------------------|
| <i>widgets</i> | Widgets module init. |
|----------------|----------------------|

Submodules

| | |
|-------------------------|---|
| <i>plotly_dash</i> | Module for dash plotly. |
| <i>plotly_interface</i> | Plotly backend interface for visualization. |

The widgets package

Summary

Submodules

| | |
|-------------------------|---|
| <i>button_manager</i> | Module for button management. |
| <i>dropdown_manager</i> | Module for dropdown management in Plotly figures. |

The `button_manager.py` module

Summary

Classes

| | |
|----------------------|---|
| <i>ButtonManager</i> | Class to manage buttons in a Plotly figure. |
|----------------------|---|

ButtonManager

class `ansys.tools.visualization_interface.backends.plotly.widgets.button_manager.ButtonManager`(*fig: plotly.graph*

Class to manage buttons in a Plotly figure.

This class allows adding buttons to a Plotly figure for various functionalities such as toggling visibility of traces, resetting the view, and custom actions.

Parameters

fig

[`go.Figure`] The Plotly figure to which buttons will be added.

Overview

Methods

| | |
|--|--|
| <code>add_button</code> | Add a button to the Plotly figure. |
| <code>show_hide_bbox_dict</code> | Generate dictionary for showing/hiding coordinate system elements. |
| <code>update_layout</code> | Update the figure layout with all controls as buttons in a single row. |
| <code>args_xy_view_button</code> | Get camera configuration for XY plane view (top-down view). |
| <code>args_xz_view_button</code> | Get camera configuration for XZ plane view (front view). |
| <code>args_yz_view_button</code> | Get camera configuration for YZ plane view (side view). |
| <code>args_iso_view_button</code> | Get camera configuration for isometric view (3D perspective). |
| <code>add_measurement_toggle_button</code> | Get configuration for measurement toggle button. |
| <code>args_projection_toggle_button</code> | Get configuration for projection toggle button. |
| <code>args_theme_toggle_button</code> | Get configuration for theme toggle button. |

Import detail

```
from ansys.tools.visualization_interface.backends.plotly.widgets.button_manager import ButtonManager
```

Method detail

`ButtonManager.add_button`(*label*: *str*, *x*: *float*, *y*: *float*, *xanchor*: *str* = 'left', *yanchor*: *str* = 'bottom', *method*: *str* = 'restyle', *args*: *List[Any]* = None, *args2*: *List[Any]* = None) → None

Add a button to the Plotly figure.

Parameters

label

[*str*] The text to display on the button.

x

[*float*] X position of the button (0-1).

y

[*float*] Y position of the button (0-1).

xanchor

[*str*, optional] X anchor point for the button, by default “left”.

yanchor

[*str*, optional] Y anchor point for the button, by default “bottom”.

method

[*str*, optional] The method to call when the button is clicked. Options include: ‘restyle’, ‘relayout’, ‘update’, ‘animate’, by default ‘restyle’.

args

[*List[Any]*, optional] Arguments to pass to the method when the button is clicked, by default None.

args2

[*List[Any]*, optional] Secondary arguments for toggle functionality, by default None.

`ButtonManager.show_hide_bbox_dict`(*toggle*: *bool* = True)

Generate dictionary for showing/hiding coordinate system elements.

Parameters

toggle

[**bool**, optional] Whether to show (True) or hide (False) the coordinate system, by default True.

Returns**dict**

Dictionary with coordinate system visibility settings.

`ButtonManager.update_layout()` → **None**

Update the figure layout with all controls as buttons in a single row.

This method builds buttons using the configuration methods and any additional buttons that were added via `add_button()`.

`ButtonManager.args_xy_view_button(label: str = 'XY View', x: float = 0.02, y: float = 1.02)` → **dict**

Get camera configuration for XY plane view (top-down view).

Parameters**label**

[**str**, optional] The text to display on the button, by default “XY View”.

x

[**float**, optional] X position of the button (0-1), by default 0.02.

y

[**float**, optional] Y position of the button (0-1), by default 1.02.

Returns**dict**

Camera configuration for XY plane view.

`ButtonManager.args_xz_view_button(label: str = 'XZ View', x: float = 0.02, y: float = 1.02)` → **dict**

Get camera configuration for XZ plane view (front view).

Parameters**label**

[**str**, optional] The text to display on the button, by default “XZ View”.

x

[**float**, optional] X position of the button (0-1), by default 0.02.

y

[**float**, optional] Y position of the button (0-1), by default 1.02.

Returns**dict**

Camera configuration for XZ plane view.

`ButtonManager.args_yz_view_button(label: str = 'YZ View', x: float = 0.02, y: float = 1.02)` → **dict**

Get camera configuration for YZ plane view (side view).

Parameters**label**

[**str**, optional] The text to display on the button, by default “YZ View”.

x

[**float**, optional] X position of the button (0-1), by default 0.02.

y
[float, optional] Y position of the button (0-1), by default 1.02.

Returns

dict
Camera configuration for YZ plane view.

`ButtonManager.args_iso_view_button(label: str = 'ISO View', x: float = 0.02, y: float = 1.02) → dict`
Get camera configuration for isometric view (3D perspective).

Parameters

label
[str, optional] The text to display on the button, by default “ISO View”.

x
[float, optional] X position of the button (0-1), by default 0.02.

y
[float, optional] Y position of the button (0-1), by default 1.02.

Returns

dict
Camera configuration for isometric view.

`ButtonManager.add_measurement_toggle_button(label: str = 'Toggle Measurement', x: float = 0.02, y: float = 0.87) → Tuple[dict, dict]`

Get configuration for measurement toggle button.

Parameters

label
[str, optional] The text to display on the button, by default “Toggle Measurement”.

x
[float, optional] X position of the button (0-1), by default 0.02.

y
[float, optional] Y position of the button (0-1), by default 0.87.

Returns

Tuple[dict, dict]
Tuple containing (enable_measurement_config, disable_measurement_config).

`ButtonManager.args_projection_toggle_button() → Tuple[dict, dict]`

Get configuration for projection toggle button.

Parameters

label
[str, optional] The text to display on the button, by default “Toggle Projection”.

x
[float, optional] X position of the button (0-1), by default 0.14.

y
[float, optional] Y position of the button (0-1), by default 1.02.

Returns

Tuple[dict, dict]
Tuple containing (perspective_projection_config, orthographic_projection_config).

```
ButtonManager.args_theme_toggle_button(label: str = 'Toggle Theme', x: float = 0.32, y: float = 1.02) →  
    Tuple[dict, dict]
```

Get configuration for theme toggle button.

Parameters

label

[str, optional] The text to display on the button, by default “Toggle Theme”.

x

[float, optional] X position of the button (0-1), by default 0.22.

y

[float, optional] Y position of the button (0-1), by default 1.02.

Returns

Tuple[dict, dict]

Tuple containing (light_theme_config, dark_theme_config).

Description

Module for button management.

The dropdown_manager.py module

Summary

Classes

| | |
|----------------------------|--|
| <i>DashDropdownManager</i> | Class to manage dropdown menus in a Plotly figure. |
|----------------------------|--|

DashDropdownManager

```
class ansys.tools.visualization_interface.backends.plotly.widgets.dropdown_manager.DashDropdownManager(  
    /
```

Class to manage dropdown menus in a Plotly figure.

This class allows adding dropdown menus to a Plotly figure for controlling mesh visibility and other properties.

Parameters

fig

[go.Figure] The Plotly figure to which dropdowns will be added.

Overview

Methods

| | |
|---|---|
| <code>add_mesh_name</code> | Add a mesh name to track for dropdown functionality. |
| <code>get_mesh_names</code> | Get the list of tracked mesh names. |
| <code>get_visibility_args_for_meshes</code> | Get visibility arguments for showing only specified meshes. |
| <code>clear</code> | Clear all tracked mesh names. |

Import detail

```
from ansys.tools.visualization_interface.backends.plotly.widgets.dropdown_manager import _  
↳ DashDropdownManager
```

Method detail

`DashDropdownManager.add_mesh_name(name: str) → None`

Add a mesh name to track for dropdown functionality.

Parameters

name

[str] The name of the mesh to track.

`DashDropdownManager.get_mesh_names() → List[str]`

Get the list of tracked mesh names.

Returns

List[str]

List of mesh names.

`DashDropdownManager.get_visibility_args_for_meshes(visible_mesh_names: List[str]) → Dict[str, Any]`

Get visibility arguments for showing only specified meshes.

Parameters

visible_mesh_names

[List[str]] List of mesh names that should be visible.

Returns

Dict[str, Any]

Arguments for restyle method to set mesh visibility.

`DashDropdownManager.clear() → None`

Clear all tracked mesh names.

Description

Module for dropdown management in Plotly figures.

Description

Widgets module init.

The plotly_dash.py module

Summary

Classes

| | |
|--------------------------|--|
| <i>PlotlyDashBackend</i> | Plotly Dash interface for visualization. |
|--------------------------|--|

PlotlyDashBackend

```
class ansys.tools.visualization_interface.backends.plotly.plotly_dash.PlotlyDashBackend(app:
                                                                    dash.Dash
                                                                    =
                                                                    None)
```

Bases: `ansys.tools.visualization_interface.backends.plotly.plotly_interface.PlotlyBackend`

Plotly Dash interface for visualization.

Overview

Methods

| | |
|---------------------------------|--|
| <code>plot</code> | Plot a single object using Plotly and track mesh names for dropdown. |
| <code>create_dash_layout</code> | Create the Dash layout with optional dropdown for mesh visibility. |
| <code>show</code> | Render the Plotly scene. |

Properties

| | |
|-------------------------------|--|
| <code>dropdown_manager</code> | Get the dropdown manager for this backend. |
|-------------------------------|--|

Import detail

```
from ansys.tools.visualization_interface.backends.plotly.plotly_dash import PlotlyDashBackend
```

Property detail

property `PlotlyDashBackend.dropdown_manager`: `ansys.tools.visualization_interface.backends.plotly.widgets.dropdown_manager.DashDropdownManager`

Get the dropdown manager for this backend.

Returns

DashDropdownManager

The dropdown manager instance.

Method detail

`PlotlyDashBackend.plot`(*plottable_object*, *name*: `str = None`, ***plotting_options*) → `None`

Plot a single object using Plotly and track mesh names for dropdown.

Parameters

plottable_object

[Any] The object to plot.

name

[`str`, optional] Name of the mesh for labeling in Plotly.

plotting_options

[`dict`] Additional plotting options.

PlotlyDashBackend.**create_dash_layout**() → dash.html.Div

Create the Dash layout with optional dropdown for mesh visibility.

Returns

html.Div

The Dash layout component.

PlotlyDashBackend.**show**(*plottable_object=None*, *screenshot: str = None*, *name_filter=None*, ***kwargs*) → plotly.graph_objects.Figure | None

Render the Plotly scene.

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.

kwargs

[dict] Additional options the selected backend accepts.

Returns

Union[go.Figure, None]

The figure of the plot if in doc building environment. Else, None.

Description

Module for dash plotly.

The plotly_interface.py module

Summary

Classes

| | |
|----------------------|-------------------------------------|
| <i>PlotlyBackend</i> | Plotly interface for visualization. |
|----------------------|-------------------------------------|

PlotlyBackend

class ansys.tools.visualization_interface.backends.plotly.plotly_interface.**PlotlyBackend**

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

Plotly interface for visualization.

Overview

Methods

| | |
|------------------|-------------------------------------|
| <i>plot_iter</i> | Plot multiple objects using Plotly. |
| <i>plot</i> | Plot a single object using Plotly. |
| <i>show</i> | Render the Plotly scene. |

Properties

| |
|--|
| <i>layout</i> Get the current layout of the Plotly figure. |
|--|

Import detail

```
from ansys.tools.visualization_interface.backends.plotly.plotly_interface import _  
↳ PlotlyBackend
```

Property detail

property `PlotlyBackend.layout`: **Any**

Get the current layout of the Plotly figure.

Returns

Any

The current layout of the Plotly figure.

Method detail

`PlotlyBackend.plot_iter(plotting_list: Iterable[Any])` → **None**

Plot multiple objects using Plotly.

Parameters

plotting_list

[**Iterable**[**Any**]] An iterable of objects to plot.

`PlotlyBackend.plot(plottable_object: pyvista.PolyData | pyvista.MultiBlock | ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot | plotly.graph_objects.Mesh3d, name: str = None, **plotting_options)` → **None**

Plot a single object using Plotly.

Parameters

plottable_object

[**Union**[*PolyData*, *pv.MultiBlock*, *MeshObjectPlot*, *go.Mesh3d*]] The object to plot. Can be a PyVista *PolyData*, *MultiBlock*, a *MeshObjectPlot*, or a Plotly *Mesh3d*.

plotting_options

[**dict**] Additional plotting options.

name

[**str**, **optional**] Name of the mesh for labeling in Plotly. Overrides the name from *MeshObjectPlot* if provided.

`PlotlyBackend.show(plottable_object=None, screenshot: str = None, name_filter=None, **kwargs)` → **plotly.graph_objects.Figure** | **None**

Render the Plotly scene.

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.

kwargs

[[dict](#)] Additional options the selected backend accepts.

Returns

[Union](#)[[go.Figure](#), [None](#)]

The figure of the plot if in doc building environment. Else, None.

Description

Plotly backend interface for visualization.

Description

Plotly initialization.

The pyvista package**Summary****Subpackages**

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

Submodules

| | |
|--------------------------|---|
| <i>picker</i> | Module for managing picking and hovering of objects in a PyVista plotter. |
| <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 trame 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>dark_mode</i> | Provides the dark mode button widget for the PyAnsys plotter. |
| <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>pick_rotation_center</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,
                                                                                    but-
                                                                                    ton_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

| |
|--|
| <i>callback</i> Get the functionality of the button, which is implemented by subclasses. |
|--|

Methods

| |
|--|
| <i>update</i> Assign the image that represents the button. |
|--|

Attributes

| |
|----------------------|
| <i>button_config</i> |
|----------------------|

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 dark_mode.py module

Summary

Classes

DarkModeButton Provides the dark mode widget for the Visualization Interface Tool Plotter class.

DarkModeButton

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.dark_mode.DarkModeButton(plotter: an-
sys.tools.visualization_interface.backends.pyvista.widgets.widget.PlotterWidget, dark_mode: bool
= False)
```

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

Provides the dark mode widget for the Visualization Interface Tool Plotter class.

Parameters

plotter_helper

[PlotterHelper] Plotter to add the dark mode widget to.

dark_mode

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

Overview

Methods

| | |
|-----------------------|--|
| <code>callback</code> | Remove or add the dark mode widget actor upon click. |
| <code>update</code> | Define the dark mode widget button parameters. |

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.dark_mode import DarkModeButton
```

Method detail

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

Remove or add the dark mode widget actor upon click.

Parameters

`state`

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

`DarkModeButton.update()` → None

Define the dark mode widget button parameters.

Description

Provides the dark mode button widget for the PyAnsys plotter.

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

callback Move the camera in the direction defined by the button.

Attributes

direction

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(*, ...)`

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, 230))`

`CameraPanDirection.XDOWN = (1, 'downarrow', (5, 190))`

`CameraPanDirection.YUP = (2, 'upyarrow', (35, 230))`

`CameraPanDirection.YDOWN = (3, 'downarrow', (35, 190))`

`CameraPanDirection.ZUP = (4, 'upzarrow', (65, 230))`

`CameraPanDirection.ZDOWN = (5, 'downarrow', (65, 190))`

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.visualization_interface.backends.pyvista.widgets.plotter.Plotter, 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

| | |
|-----------------|---|
| <i>callback</i> | Remove or add the hide widget actor upon click. |
| <i>update</i> | 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.visualizat
    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

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

Attributes

| |
|-----------------------------|
| <code>plotter_helper</code> |
|-----------------------------|

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

| | |
|-------------------------------|---|
| <code>MeshSliderWidget</code> | 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

| |
|-----------------------------|
| <code>plotter_helper</code> |
|-----------------------------|

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 `pick_rotation_center.py` module

Summary

Classes

| | |
|----------------------------------|--|
| <code>PickRotCenterButton</code> | Provides the pick rotation center widget for the Visualization Interface Tool Plotter class. |
|----------------------------------|--|

PickRotCenterButton

```
class ansys.tools.visualization_interface.backends.pyvista.widgets.pick_rotation_center.PickRotCenterBu
```

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

Provides the pick rotation center widget for the Visualization Interface Tool `Plotter` class.

Parameters

plotter_helper

[`PlotterHelper`] Plotter to add the pick rotation center widget to.

dark_mode

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

Overview

Methods

| | |
|-----------------------|---|
| <code>callback</code> | Remove or add the pick rotation center widget actor upon click. |
| <code>update</code> | Define the measurement widget button parameters. |

Attributes

`plotter_helper`

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.widgets.pick_rotation_center_
↪ import PickRotCenterButton
```

Attribute detail

`PickRotCenterButton.plotter_helper`

Method detail

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

Remove or add the pick rotation center widget actor upon click.

Parameters

state

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

`PickRotCenterButton.update()` → `None`

Define the measurement 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

| | |
|-------------------------------|---|
| <code>ScreenshotButton</code> | Provides the screenshot widget for the Visualization Interface Tool <code>Plotter</code> 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

| | |
|-----------------------|--|
| <code>callback</code> | Remove or add the screenshot widget actor upon click. |
| <code>update</code> | 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, 280))
ViewDirection.XYMINUS = (1, '-xy', (5, 311))
ViewDirection.XZPLUS = (2, '+xz', (5, 342))
ViewDirection.XZMINUS = (3, '-xz', (5, 373))
ViewDirection.YZPLUS = (4, '+yz', (5, 404))
ViewDirection.YZMINUS = (5, '-yz', (5, 435))
ViewDirection.ISOMETRIC = (6, 'isometric', (5, 466))
```

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 `picker.py` module**Summary**

Classes

| | |
|-----------------------------|---|
| <code>AbstractPicker</code> | Abstract base class for pickers. |
| <code>Picker</code> | Class to manage picking and hovering of objects in the plotter. |

AbstractPicker

```
class ansys.tools.visualization_interface.backends.pyvista.picker.AbstractPicker(plotter_backend:
an-
sys.tools.visualization_interfa
**kwargs)
```

Bases: `abc.ABC`

Abstract base class for pickers.

Overview

Abstract methods

| | |
|------------------------------------|---|
| <code>pick_select_object</code> | Determine actions to take when an object is selected. |
| <code>pick_unselect_object</code> | Determine actions to take when an object is unselected. |
| <code>hover_select_object</code> | Determine actions to take when an object is hovered over. |
| <code>hover_unselect_object</code> | Determine actions to take when an object is unhovered. |

Properties

| | |
|--------------------------|--|
| <code>picked_dict</code> | Return the dictionary of picked objects. |
|--------------------------|--|

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.picker import AbstractPicker
```

Property detail

property `AbstractPicker.picked_dict`: `dict`

Abstractmethod

Return the dictionary of picked objects.

Method detail

abstractmethod `AbstractPicker.pick_select_object`(*custom_object*: `an-`
`sys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPl`
`| an-`
`sys.tools.visualization_interface.types.edge_plot.EdgePlot,`
pt: `numpy.ndarray`) \rightarrow `None`

Determine actions to take when an object is selected.

abstractmethod `AbstractPicker.pick_unselect_object`(*custom_object*: `an-`
`sys.tools.visualization_interface.types.mesh_object_plot.MeshObject`
| `an-`
`sys.tools.visualization_interface.types.edge_plot.EdgePlot`)
→ `None`

Determine actions to take when an object is unselected.

abstractmethod `AbstractPicker.hover_select_object`(*custom_object*: `an-`
`sys.tools.visualization_interface.types.mesh_object_plot.MeshObject`
| `an-`
`sys.tools.visualization_interface.types.edge_plot.EdgePlot`,
pt: `numpy.ndarray`) → `None`

Determine actions to take when an object is hovered over.

abstractmethod `AbstractPicker.hover_unselect_object`(*custom_object*: `an-`
`sys.tools.visualization_interface.types.mesh_object_plot.MeshObject`
| `an-`
`sys.tools.visualization_interface.types.edge_plot.EdgePlot`)
→ `None`

Determine actions to take when an object is unhovered.

Picker

class `ansys.tools.visualization_interface.backends.pyvista.picker.Picker`(*plotter_backend*: `an-`
`sys.tools.visualization_interface.backends.pyvista.plotter.Plotter`
| *plot_picked_names*:
`bool = True`)

Bases: `AbstractPicker`

Class to manage picking and hovering of objects in the plotter.

This class is responsible for managing the selection and deselection of objects in the plotter, both through direct picking and hovering. It keeps track of the currently selected and hovered objects, and provides methods to select and unselect them.

Parameters

plotter_backend

[`Plotter`] The plotter instance to which this picker is attached.

plot_picked_names

[`bool`, optional] Whether to display the names of picked objects in the plotter. Defaults to `True`.

Overview

Methods

| | |
|------------------------------------|---|
| <code>pick_select_object</code> | Add actor to picked list and add label if required. |
| <code>pick_unselect_object</code> | Remove actor from picked list and remove label if required. |
| <code>hover_select_object</code> | Add label to hovered object if required. |
| <code>hover_unselect_object</code> | Remove all hover labels from the scene. |

Properties

| | |
|--------------------------|--|
| <code>picked_dict</code> | Return the dictionary of picked objects. |
|--------------------------|--|

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.picker import Picker
```

Property detail

property `Picker.picked_dict`: `dict`

Return the dictionary of picked objects.

Method detail

`Picker.pick_select_object`(*custom_object*:
 `ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot |`
 `ansys.tools.visualization_interface.types.edge_plot.EdgePlot`, *pt*: `numpy.ndarray`)
 → `None`

Add actor to picked list and add label if required.

Parameters

custom_object

[Union[MeshObjectPlot, EdgePlot]] The object to be selected.

pt

[`np.ndarray`] The point where the object was picked.

`Picker.pick_unselect_object`(*custom_object*:
 `ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot |`
 `ansys.tools.visualization_interface.types.edge_plot.EdgePlot`) → `None`

Remove actor from picked list and remove label if required.

Parameters

custom_object

[Union[MeshObjectPlot, EdgePlot]] The object to be unselected.

`Picker.hover_select_object`(*custom_object*:
 `ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot |`
 `ansys.tools.visualization_interface.types.edge_plot.EdgePlot`, *actor*:
 `pyvista.Actor`) → `None`

Add label to hovered object if required.

Parameters

custom_object

[Union[MeshObjectPlot, EdgePlot]] The object to be hovered over.

actor

[`vtkActor`] The actor corresponding to the hovered object.

`Picker.hover_unselect_object`()

Remove all hover labels from the scene.

Description

Module for managing picking and hovering of objects in a PyVista 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_frame: bool
    |
    None
    =
    None,
    al-
    low_picking: bool
    |
    None
    =
    False,
    al-
    low_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,
    cus-
    tom_picker: an-
    sys.tools.visualization_interface.backends.pyvista.pyvista.PyVistaBackendInterface
    =
    None,
    cus-
    tom_picker_kwargs: Dict[str, Any]
    |
    None
    =
    None,
    **plot-
    ter_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_trame**
[Optional[bool], default: `None`] Whether to activate the usage of the trame 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.
- custom_picker**
[AbstractPicker, default: `None`] Custom picker class that extends the `AbstractPicker` class.
- custom_picker_kwargs**
[Optional[Dict[str, Any]], default: `None`] Keyword arguments to pass to the custom picker class.

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>picker_callback</code> | Define the callback for the element picker. |
| <code>hover_callback</code> | Define the callback for the element hover. |
| <code>focus_point_selection</code> | Focus the camera on a selected actor. |
| <code>compute_edge_object_map</code> | Compute the mapping between plotter actors and <code>EdgePlot</code> objects. |
| <code>enable_picking</code> | Enable picking capabilities in the plotter. |
| <code>enable_set_focus_center</code> | Enable setting the focus of the camera to the picked point. |
| <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>disable_center_focus</code> | Disable setting the focus of the camera to the picked point. |
| <code>show</code> | Plot and show any PyAnsys object. |
| <code>show_plotter</code> | Show the plotter or start the <code>trame</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.**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.**focus_point_selection**(actor: *pyvista.Actor*) → None

Focus the camera on a selected actor.

Parameters

actor

[*Actor*] Actor to focus the camera on.

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_set_focus_center**()

Enable setting the focus of the camera to the picked point.

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.**disable_center_focus**()

Disable setting the focus of the camera to the picked point.

PyVistaBackendInterface.**show**(plottable_object: Any = None, screenshot: *str* | None = None, view_2d: Dict = None, name_filter: *str* = None, dark_mode: *bool* = False, **kwargs: 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.

****kwargs**

[Any] Additional keyword arguments for the show or plot 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 plot.

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,
    custom_picker:
    ansys.tools.visualization_inter
    = None)
```

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 plot.

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>hide_children</code> | Hide all the children of a given <code>MeshObjectPlot</code> object. |
| <code>show_children</code> | Show all the children of a given <code>MeshObjectPlot</code> object. |
| <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, plot_children: bool = True, **plotting_options)`

Plot a generic *MeshObjectPlot* object to the scene.

Parameters

plottable_object

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

plot_children

[*bool*, default: `True`] Whether to plot the children of the object.

****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.hide_children(custom_object: an-
    sys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot)
    → None
```

Hide all the children of a given `MeshObjectPlot` object.

Parameters

custom_object

[`MeshObjectPlot`] Custom object whose children will be hidden.

```
PyVistaInterface.show_children(custom_object: an-
    sys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot)
    → None
```

Show all the children of a given `MeshObjectPlot` object.

Parameters

custom_object

[`MeshObjectPlot`] Custom object whose children will be shown.

```
PyVistaInterface.plot(plottable_object: pyvista.PolyData | pyvista.MultiBlock |
    ansys.tools.visualization_interface.types.mesh_object_plot.MeshObjectPlot |
    pyvista.UnstructuredGrid, name_filter: str = None, plot_children: bool = False,
    **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 and show keyword arguments. For allowable keyword arguments, see the `Plotter.show` and `Plotter.show` methods.

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

| |
|---|
| <code>TrameVisualizer</code> Defines the trame layout view. |
|---|

Constants

CLIENT_TYPE

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

| |
|----------------|
| <i>server</i> |
| <i>plotter</i> |

Import detail

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

Attribute detail

TrameVisualizer.**server** = None

TrameVisualizer.**plotter** = 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

| | |
|---------------------------|----------------------|
| <code>TrameService</code> | 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

| | |
|----------------------|--|
| <i>clear_plotter</i> | Clears the web view in the service. |
| <i>set_scene</i> | Sets the web view scene for the trame service. |
| <i>run</i> | Start the trame web view and the websocket services. |

Import detail

```
from ansys.tools.visualization_interface.backends.pyvista.trame_service import_
↪ TrameService
```

Method detail

TrameService.clear_plotter()

Clears the web view in the service.

TrameService.set_scene()

Sets the web view scene for the trame service.

TrameService.run()

Start the trame web view and the websocket services.

Description

Trame service module.

Description

Provides interfaces.

Description

Provides interfaces.

The types package

Summary

Submodules

| | |
|-------------------------------|--------------------------------------|
| <code>edge_plot</code> | Provides the edge type for plotting. |
| <code>mesh_object_plot</code> | Provides the MeshObjectPlot class. |

The edge_plot.py module

Summary

Classes

| | |
|-----------------------|---|
| <code>EdgePlot</code> | 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 |  
                                                                    plotly.graph_objects.Mesh3d,  
                                                                    edge_object: Any, parent:  
                                                                    Any = None)
```

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

Parameters

actor

[Union[[Actor](#), [Mesh3d](#)]] 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

| | |
|--------------------------|------------------------------------|
| <code>actor</code> | PyVista actor of the object. |
| <code>edge_object</code> | PyAnsys edge. |
| <code>parent</code> | Parent PyAnsys object of the edge. |
| <code>name</code> | 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

| | |
|-----------------------------|--|
| <code>MeshObjectPlot</code> | 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
        |
        plotly.graph_objects.Mesh3d,
    actor:
        pyvista.Actor
        = None,
    edges:
        List[ansys.tools.visualization_
        = None,
    children:
        List[MeshObjectPlot]
        = None,
    parent:
        MeshOb-
        jectPlot =
        None)

```

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

Overview

Methods

| | |
|------------------------|---|
| <code>add_child</code> | Set a child MeshObjectPlot to the current object. |
|------------------------|---|

Properties

| | |
|----------------------------|--|
| <code>parent</code> | Get the parent MeshObjectPlot of the current object. |
| <code>mesh</code> | Mesh of the object in PyVista format. |
| <code>custom_object</code> | Custom object. |
| <code>actor</code> | PyVista actor of the object in the plotter. |
| <code>edges</code> | Edges of the object. |
| <code>name</code> | Name of the object. |
| <code>mesh_type</code> | Type of the mesh. |

Import detail

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

Property detail

property MeshObjectPlot.**parent**: *MeshObjectPlot*

Get the parent MeshObjectPlot of the current object.

This method is used to set a parent MeshObjectPlot to the current object. It is useful when the custom object has a hierarchical structure, and the consumer library wants to relate the parent objects with their meshes.

Parameters**parent**

[MeshObjectPlot] Parent MeshObjectPlot to be set.

property MeshObjectPlot.**mesh:** `pyvista.PolyData` | `pyvista.MultiBlock` | `plotly.graph_objects.Mesh3d`

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.

property MeshObjectPlot.**mesh_type:** `Type`

Type of the mesh.

Returns

`type`

Type of the mesh.

Method detail

MeshObjectPlot.add_child(child: MeshObjectPlot)

Set a child MeshObjectPlot to the current object.

This method is used to set a child MeshObjectPlot to the current object. It is useful when the custom object has a hierarchical structure, and the consumer library wants to relate the child objects with their meshes.

Parameters

child

[MeshObjectPlot] Child MeshObjectPlot to be set.

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

| | |
|---------------|----------------------|
| <i>normal</i> | Normal of the plane. |
| <i>origin</i> | 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

logger

SingletonType

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

Bases: `type`

Provides the singleton helper class for the logger.

Overview

Special methods

| | |
|-----------------|---|
| <i>__call__</i> | 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

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

Plotter

class `ansys.tools.visualization_interface.plotter.Plotter`(*backend: ansys.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_iter</code> | Plots multiple objects using the specified backend. |
| <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_iter(plotting_list: List, **plotting_options)`

Plots multiple objects using the specified backend.

Parameters

plotting_list

[List] List of objects to plot.

plotting_options

[dict] Additional plotting options.

`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, **kwargs) → List`

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.

kwargs

[[dict](#)] Additional options the selected backend accepts.

Returns

List

List of picked objects.

Description

Module for the Plotter class.

4.1.2 Description

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

4.1.3 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 shows 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.

BASIC PLOTLY USAGE EXAMPLES

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

ADVANCED USAGE EXAMPLES

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

8.1 Basic usage examples

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

8.1.1 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)

8.1.2 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

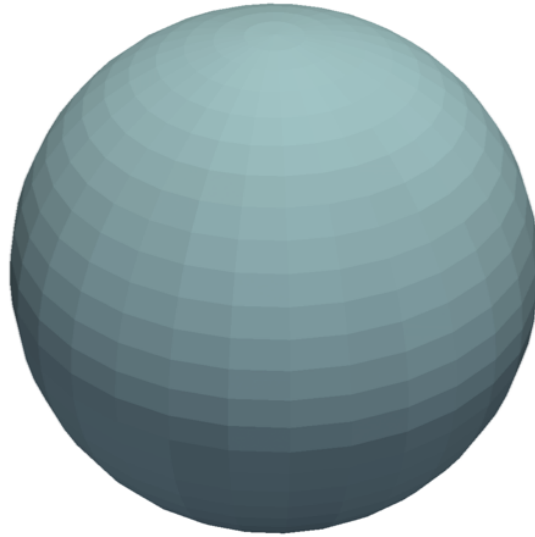
```
cube = pv.Cube()
pv_backend = PyVistaBackend(use_qt=True, show_qt=True)
pl = Plotter(backend=pv_backend)
pl.plot(cube)
pl.backend.enable_widgets()
pv_backend.scene.show()
```

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 5.483 seconds)

8.1.3 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()
```

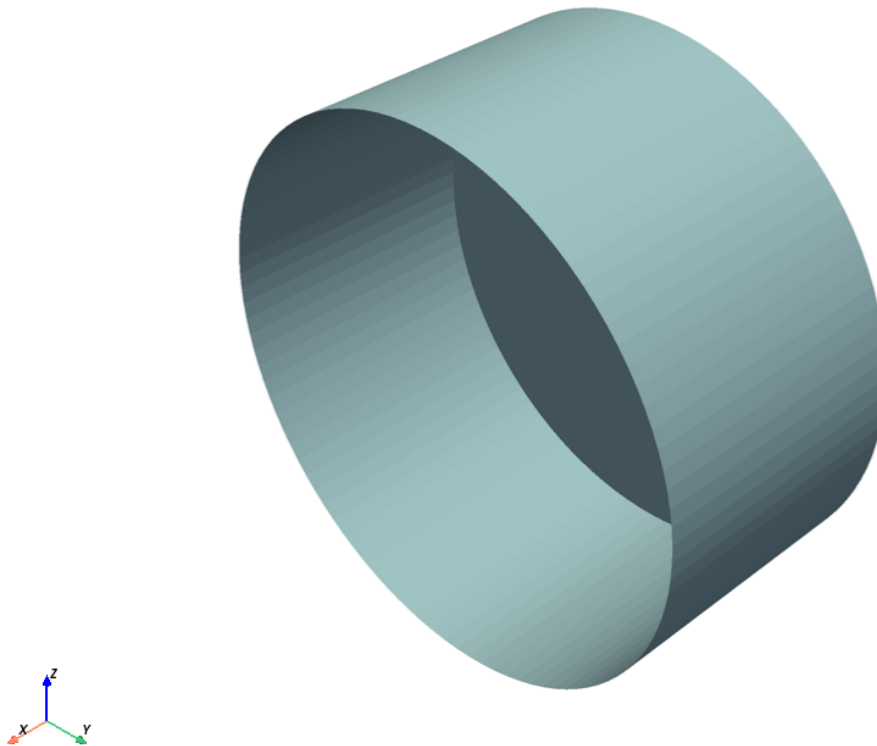
Create a plotter and clip the mesh

```
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 0.327 seconds)

8.1.4 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

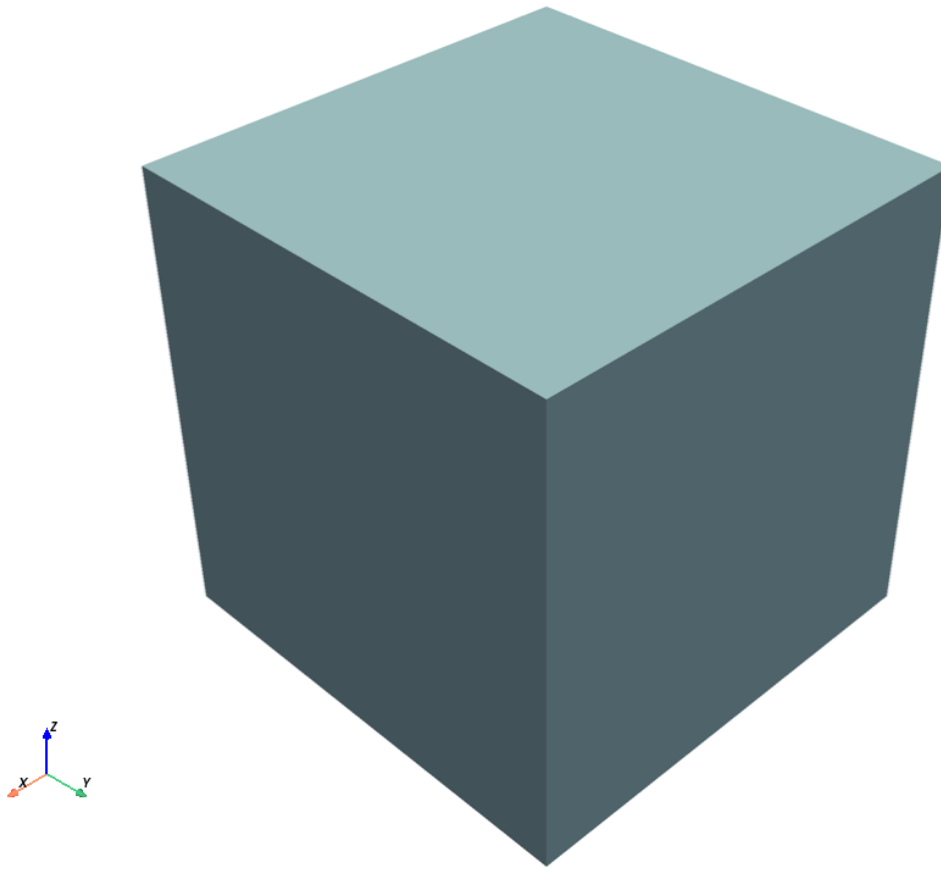
mesh_object = MeshObjectPlot(custom_object, custom_object.get_mesh())
```

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 0.343 seconds)

8.1.5 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()
```

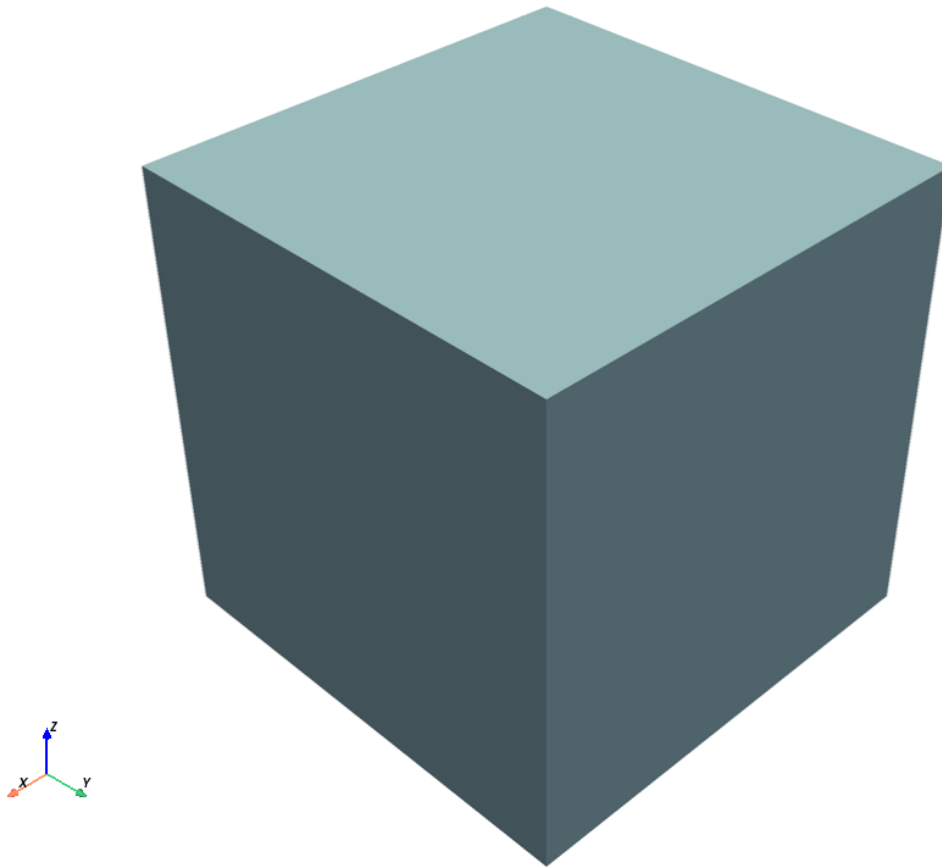
(continues on next page)

(continued from previous page)

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

# 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
```

(continues on next page)

(continued from previous page)

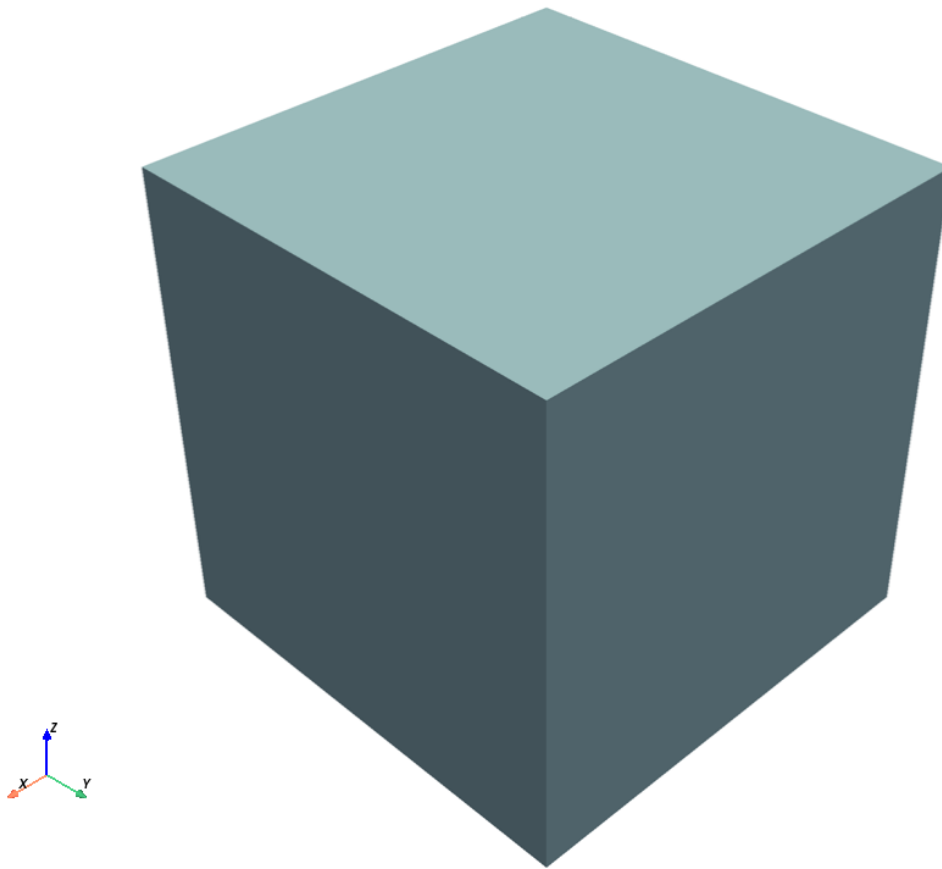
```
mesh = pv.Cube()

# Create a plotter
pl = Plotter()

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

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

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

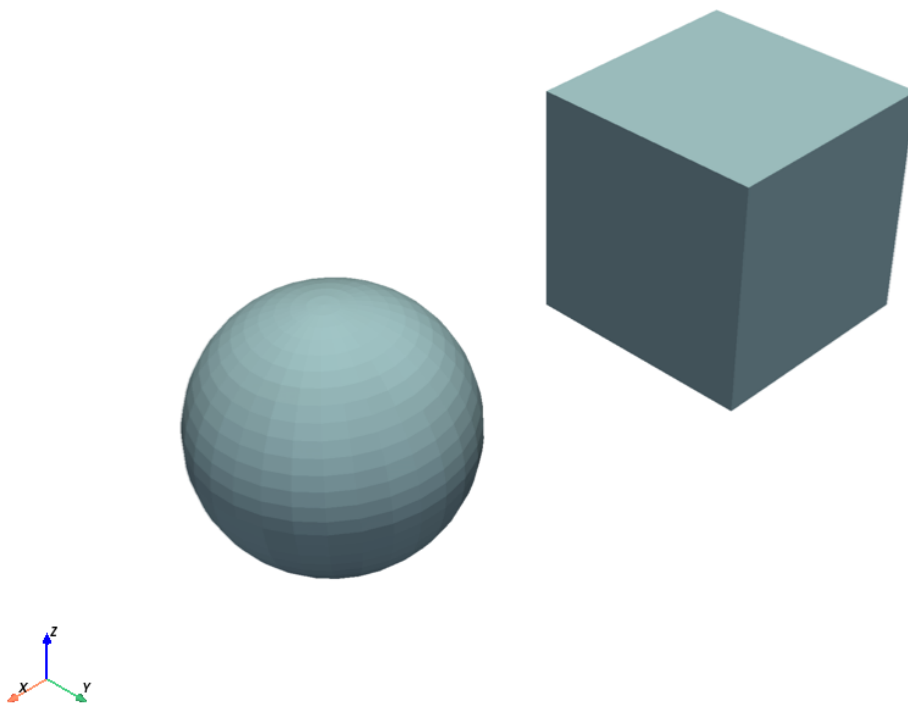
from ansys.tools.visualization_interface import Plotter

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

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

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

Static Scene



Interactive Scene

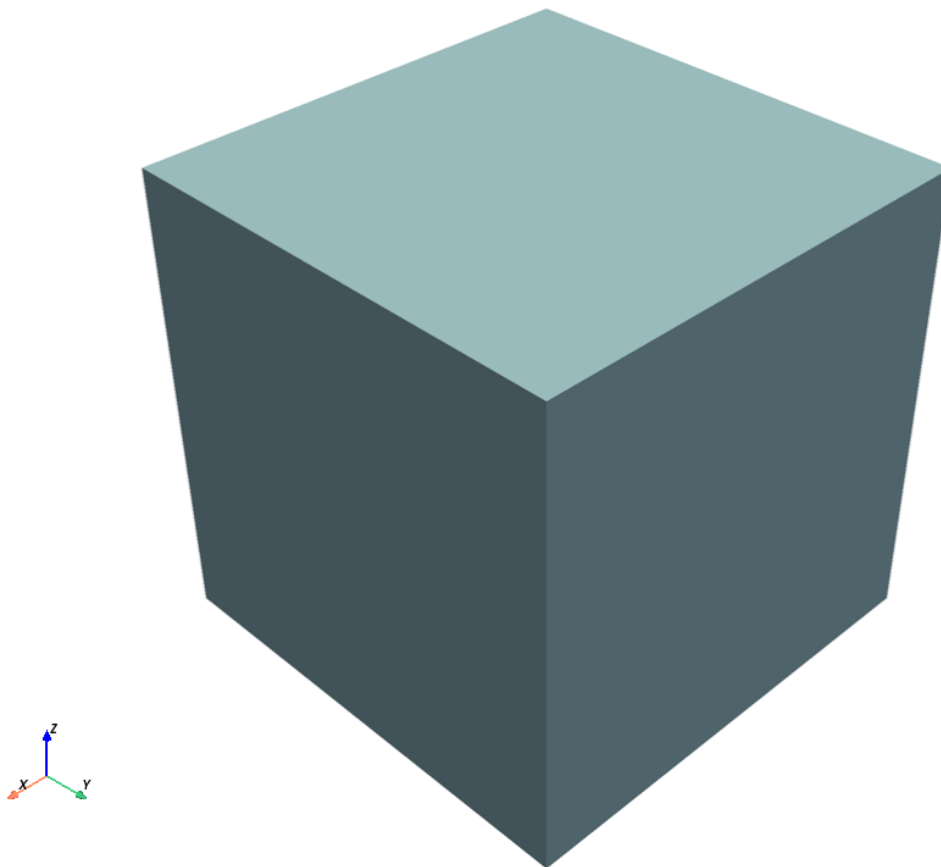
[]

Total running time of the script: (0 minutes 0.996 seconds)

8.1.6 MeshObjectPlot tree structure

This example shows how to add a tree structure of MeshObjectPlot to the plotter.

Static Scene



Interactive Scene

[]

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

(continues on next page)

(continued from previous page)

```

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 objects
custom_cube = CustomObject()
custom_cube.name = "CustomCube"

custom_sphere = CustomObject()
custom_sphere.mesh = pv.Sphere(center=(0, 0, 5))
custom_sphere.name = "CustomSphere"

custom_sphere1 = CustomObject()
custom_sphere1.mesh = pv.Sphere(center=(5, 0, 5))
custom_sphere1.name = "CustomSphere"

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())
mesh_object_sphere1 = MeshObjectPlot(custom_sphere1, custom_sphere1.get_mesh())

mesh_object_cube.add_child(mesh_object_sphere)
mesh_object_sphere.add_child(mesh_object_sphere1)

pl = Plotter()
pl.plot(mesh_object_cube, plot_children=True)

pl.backend._pl.hide_children(mesh_object_cube)
pl.show()

```

Total running time of the script: (0 minutes 0.366 seconds)

8.1.7 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

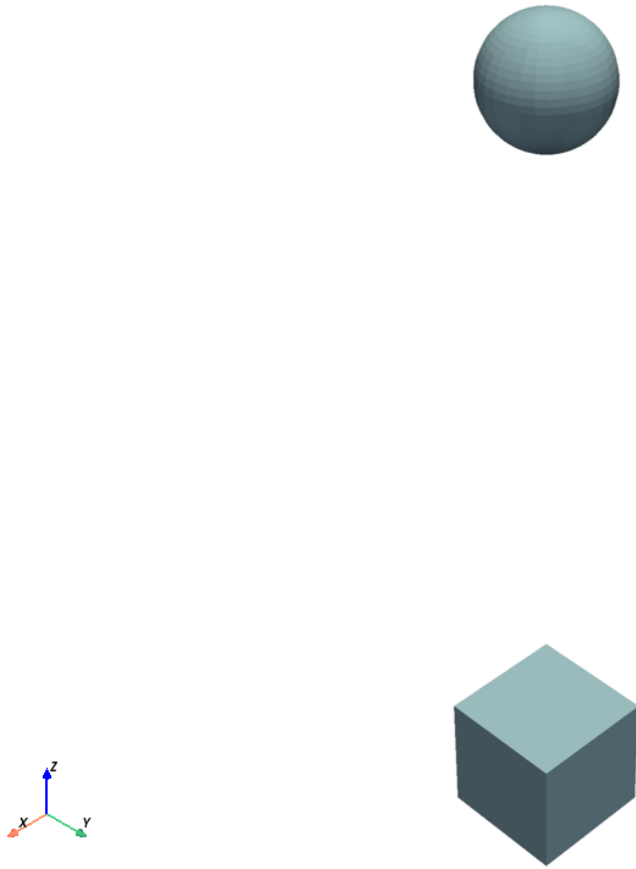
# Create an instance
mesh_object_cube = MeshObjectPlot(custom_cube, custom_cube.get_mesh())
mesh_object_sphere = MeshObjectPlot(custom_sphere, custom_sphere.get_mesh())
```

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

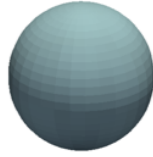
```
[ ]
```

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

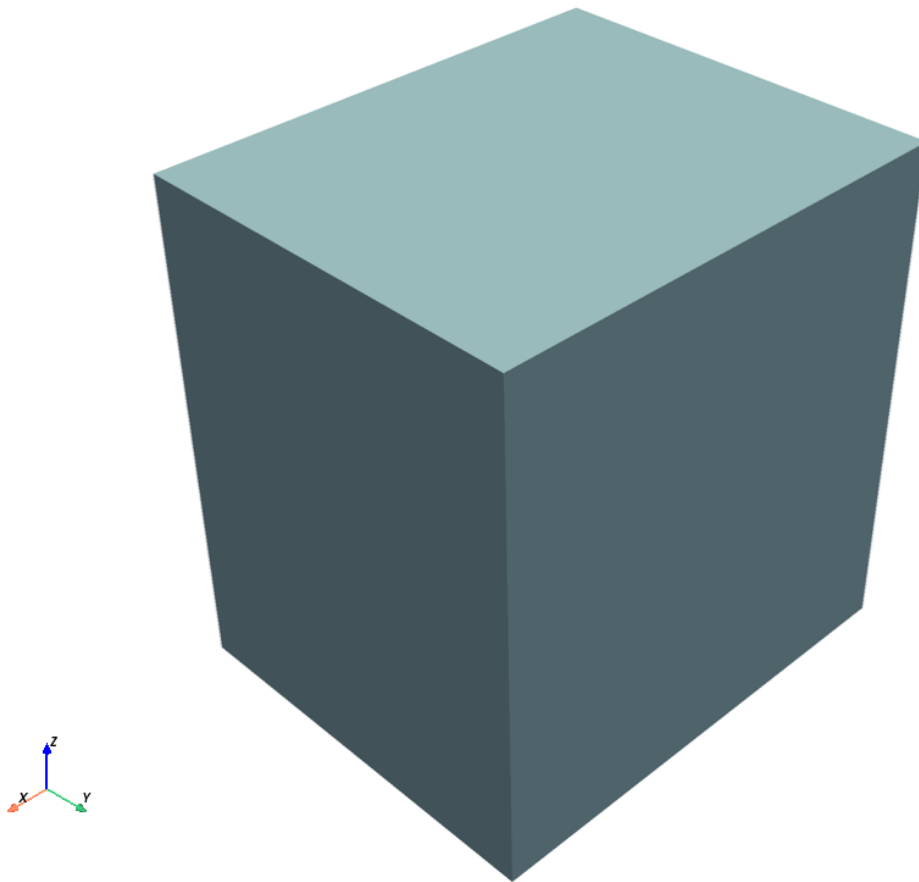
    def get_mesh(self):
        return self.mesh
```

(continues on next page)

(continued from previous page)

```
def name(self):  
    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 0.937 seconds)

8.1.8 Create custom picker

This example shows how to create a custom picker. In this case we will show how the default picker is implemented through the `AbstractPicker` class.

Import the `AbstractPicker` class

```
# Import the abstract picker class
from ansys.tools.visualization_interface.backends.pyvista.picker import AbstractPicker

# Import custom object meshes
from ansys.tools.visualization_interface.types.mesh_object_plot import MeshObjectPlot

# Import plotter and color enum
from ansys.tools.visualization_interface import Plotter
from ansys.tools.visualization_interface.utils.color import Color
```

Create a custom picker class

```
class CustomPicker(AbstractPicker):
    """Custom picker class that extends the AbstractPicker.
    This custom picker changes the color of picked objects to red and adds a label with
    the object's name.
    It also adds a label when hovering over an object.

    Parameters
    -----
    plotter_backend : Plotter
        The plotter backend to use.
    plot_picked_names : bool, optional
        Whether to plot the names of picked objects, by default True.
    label : str, optional
        Extra parameter to exemplify the usage of custom parameters.
    """
    def __init__(self, plotter_backend: "Plotter", plot_picked_names: bool = True,
    label: str = "This label: ") -> None:
        """Initialize the ``Picker`` class."""
        # Picking variables
        self._plotter_backend = plotter_backend
        self._plot_picked_names = plot_picked_names
        self._label = label

        # Map that relates PyVista actors with the added actors by the picker
        self._picker_added_actors_map = {}

        # Dictionary of picked objects in MeshObject format.
        self._picked_dict = {}

        # Map that saves original colors of the plotted objects.
        self._origin_colors = {}

        # Hovering variables
        self._added_hover_labels = []
```

(continues on next page)

(continued from previous page)

```

def pick_select_object(self, custom_object: MeshObjectPlot, pt: "np.ndarray") ->
↳None:
    """Add actor to picked list and add label if required.

    Parameters
    -----
    custom_object : MeshObjectPlot
        The object to be selected.
    pt : np.ndarray
        The point where the object was picked.
    """
    added_actors = []

    # Pick only custom objects
    if isinstance(custom_object, MeshObjectPlot):
        self._origin_colors[custom_object] = custom_object.actor.prop.color
        custom_object.actor.prop.color = Color.PICKED.value

    # Get the name for the text label
    text = custom_object.name

    # If picking names is enabled, add a label to the picked object
    if self._plot_picked_names:
        label_actor = self._plotter_backend.pv_interface.scene.add_point_labels(
            [pt],
            [self._label + text],
            always_visible=True,
            point_size=0,
            render_points_as_spheres=False,
            show_points=False,
        )
        # Add the label actor to the list of added actors
        added_actors.append(label_actor)

    # Add the picked object to the picked dictionary if not already present, to keep
↳track of it
    if custom_object.name not in self._picked_dict:
        self._picked_dict[custom_object.name] = custom_object
    # Add the picked object to the picked dictionary if not already present, to keep
↳track of it
    self._picker_added_actors_map[custom_object.actor.name] = added_actors

def pick_unselect_object(self, custom_object: MeshObjectPlot) -> None:
    """Remove actor from picked list and remove label if required.

    Parameters
    -----
    custom_object : MeshObjectPlot
        The object to be unselected.
    """
    # remove actor from picked list and from scene

```

(continues on next page)

(continued from previous page)

```

    if custom_object.name in self._picked_dict:
        self._picked_dict.pop(custom_object.name)

    # Restore original color if it was changed
    if isinstance(custom_object, MeshObjectPlot) and custom_object in self._origin_
↪ colors:
        custom_object.actor.prop.color = self._origin_colors[custom_object]

    # Remove any added actors (like labels) associated with this picked object
    if custom_object.actor.name in self._picker_added_actors_map:
        self._plotter_backend._pl.scene.remove_actor(self._picker_added_actors_
↪ map[custom_object.actor.name])
        self._picker_added_actors_map.pop(custom_object.actor.name)

def hover_select_object(self, custom_object: MeshObjectPlot, actor: "Actor") -> None:
    """Add label to hovered object if required.

    Parameters
    -----
    custom_object : MeshObjectPlot
        The object to be hovered over.
    actor : vtkActor
        The actor corresponding to the hovered object.
    """
    for label in self._added_hover_labels:
        self._plotter_backend._pl.scene.remove_actor(label)
    label_actor = self._plotter_backend._pl.scene.add_point_labels(
        [actor.GetCenter()],
        [custom_object.name],
        always_visible=True,
        point_size=0,
        render_points_as_spheres=False,
        show_points=False,
    )
    self._added_hover_labels.append(label_actor)

def hover_unselect_object(self):
    """Remove all hover labels from the scene."""
    for label in self._added_hover_labels:
        self._plotter_backend._pl.scene.remove_actor(label)

@property
def picked_dict(self) -> dict:
    """Return the dictionary of picked objects.

    Returns
    -----
    dict
        Dictionary of picked objects.
    """
    return self._picked_dict

```

Initialize the plotter backend with the custom picker

```
from ansys.tools.visualization_interface.backends.pyvista import PyVistaBackend
pl_backend = PyVistaBackend(allow_picking=True, custom_picker=CustomPicker)
```



Create a custom object with a name to be picked

```
import pyvista as pv

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"
```

Create a MeshObjectPlot instance

```
from ansys.tools.visualization_interface import MeshObjectPlot
# Create an instance
mesh_object_cube = MeshObjectPlot(custom_cube, custom_cube.get_mesh())
```

Display the plotter and interact with the object

```
pl = Plotter(backend=pl_backend)
pl.plot(mesh_object_cube)
pl.show()
```

Total running time of the script: (0 minutes 0.413 seconds)

8.2 Basic Plotly usage examples

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

8.2.1 Plain usage of the plotly dash backend

This example shows the plain usage of the Plotly Dash backend in the Visualization Interface Tool to plot different objects, including PyVista meshes, custom objects, and Plotly-specific objects.

```
from ansys.tools.visualization_interface.backends.plotly.plotly_dash import _
↳ PlotlyDashBackend
from ansys.tools.visualization_interface.types.mesh_object_plot import MeshObjectPlot
from ansys.tools.visualization_interface import Plotter
import pyvista as pv
from plotly.graph_objects import Mesh3d

# Create a plotter with the Plotly backend
pl = Plotter(backend=PlotlyDashBackend())

# Create a PyVista mesh
mesh = pv.Sphere()
mesh2 = pv.Cube(center=(2,0,0))
# Plot the mesh
pl.plot(mesh, name="Sphere")
pl.plot(mesh2, name="Cube")

# -----
# Start the server and show the plot
# -----
#
# .. code-block:: python
#
#     pl.show()
```

Total running time of the script: (0 minutes 0.827 seconds)

8.2.2 Plain usage of the plotly backend

This example shows the plain usage of the Plotly backend in the Visualization Interface Tool to plot different objects, including PyVista meshes, custom objects, and Plotly-specific objects.

```
from ansys.tools.visualization_interface.backends.plotly.plotly_interface import _
    PlotlyBackend
from ansys.tools.visualization_interface.types.mesh_object_plot import MeshObjectPlot
from ansys.tools.visualization_interface import Plotter
import pyvista as pv
from plotly.graph_objects import Mesh3d

# Create a plotter with the Plotly backend
pl = Plotter(backend=PlotlyBackend())

# Create a PyVista mesh
mesh = pv.Sphere()

# Plot the mesh
pl.plot(mesh)

# Create a PyVista MultiBlock
multi_block = pv.MultiBlock()
multi_block.append(pv.Sphere(center=(-1, -1, 0)))
multi_block.append(pv.Cube(center=(-1, 1, 0)))

# Plot the MultiBlock
pl.plot(multi_block)

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

Now create a custom object

```
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"
```

(continues on next page)

(continued from previous page)

```
# Create a MeshObjectPlot instance
mesh_object_cube = MeshObjectPlot(custom_cube, custom_cube.get_mesh())

# Plot the custom mesh object
pl.plot(mesh_object_cube)
```

Display the plotter again

Since Plotly is a web-based visualization, we can show the plot again to include the new object.

```
pl.show()
```

Add a Plotly Mesh3d object directly

```
custom_mesh3d = Mesh3d(
    x=[0, 1, 2],
    y=[0, 1, 0],
    z=[0, 0, 1],
    i=[0],
    j=[1],
    k=[2],
    color='lightblue',
    opacity=0.50
)
pl.plot(custom_mesh3d, name="CustomMesh3d")

# Show other plotly objects like Scatter3d
from plotly.graph_objects import Scatter3d

scatter = Scatter3d(
    x=[0, 1, 2],
    y=[0, 1, 0],
    z=[0, 0, 1],
    mode='markers',
    marker=dict(size=5, color='red')
)
pl.plot(scatter, name="CustomScatter3d")

pl.show()
```

Total running time of the script: (0 minutes 0.083 seconds)

8.3 Advanced usage examples

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

8.3.1 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}")
```

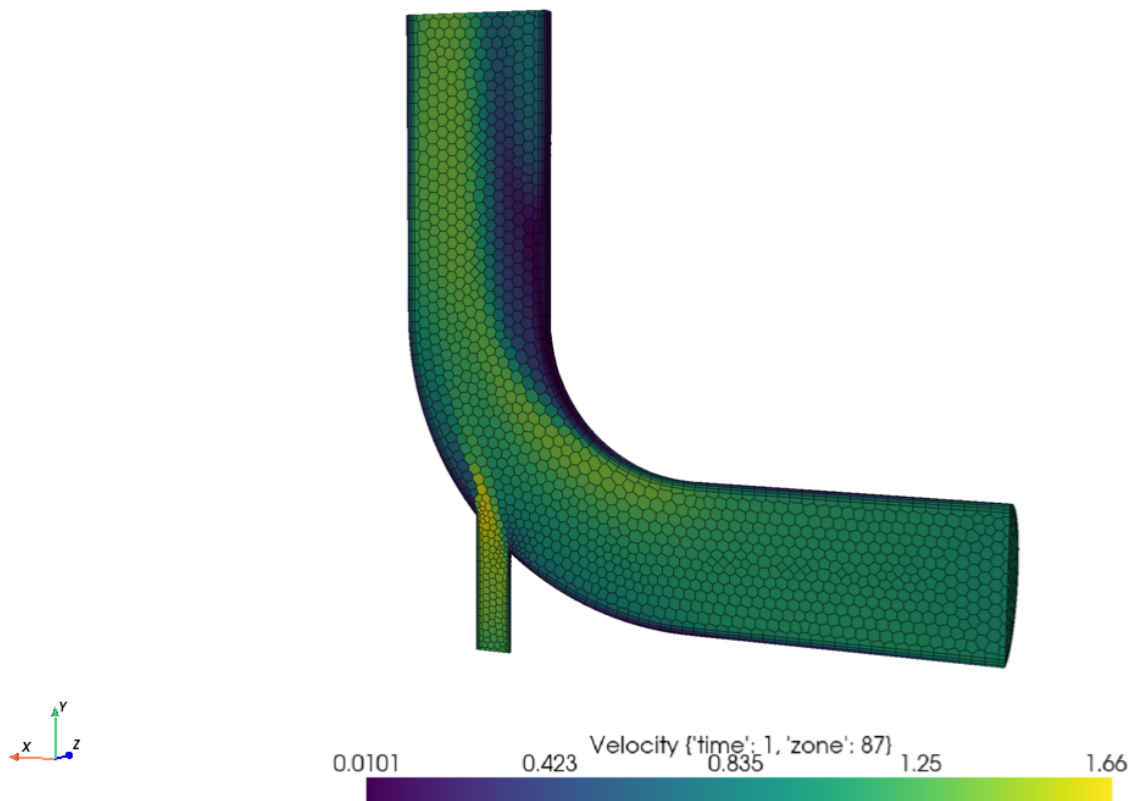


```
Field data arrays: ["Velocity {'time': 1, 'zone': 87}", "Temperature {'time': 1, 'zone': 87}"]
```

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



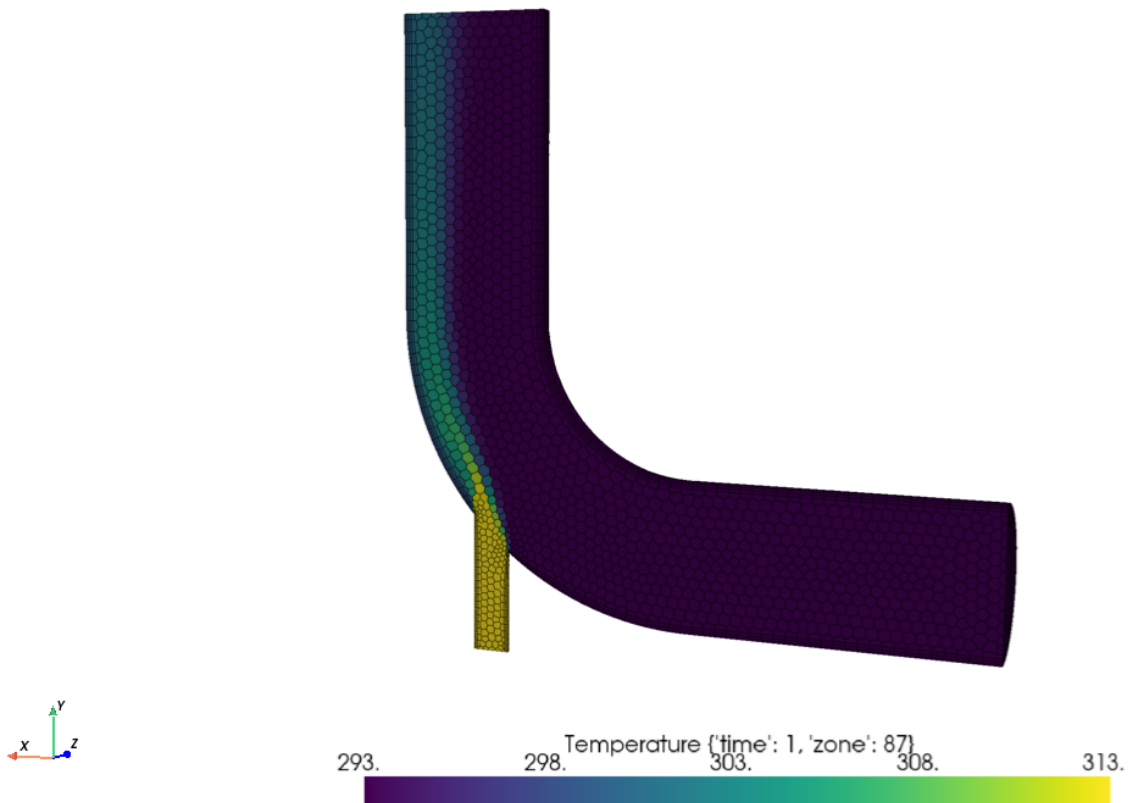
Interactive Scene

```
[]
```

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 6.850 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.

9.1 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 .
```

9.2 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
```

9.3 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
```

9.4 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
```

9.5 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.backends.pyvista.widgets`
33
`ansys.tools.visualization_interface.backends`, `ansys.tools.visualization_interface.backends.pyvista.widgets`
15 34
`ansys.tools.visualization_interface.backends.pyvista`, `ansys.tools.visualization_interface.backends.pyvista.widgets`
16 36
`ansys.tools.visualization_interface.backends.pyvista.widgets`, `ansys.tools.visualization_interface.backends.pyvista.widgets`
21 37
`ansys.tools.visualization_interface.backends.pyvista.widgets`, `ansys.tools.visualization_interface.backends.pyvista.widgets`
23 38
`ansys.tools.visualization_interface.backends.pyvista.widgets`, `ansys.tools.visualization_interface.backends.pyvista.widgets`
16 40
`ansys.tools.visualization_interface.backends.pyvista.widgets.button_manager`, `ansys.tools.visualization_interface.backends.pyvista.widgets.button_manager`
16 70
`ansys.tools.visualization_interface.backends.pyvista.widgets.dropdown_manager`, `ansys.tools.visualization_interface.backends.pyvista.widgets.dropdown_manager`
20 62
`ansys.tools.visualization_interface.backends.pyvista.widgets.edge_plot`, `ansys.tools.visualization_interface.backends.pyvista.widgets.edge_plot`
25 62
`ansys.tools.visualization_interface.backends.pyvista.widgets.mesh_object_plot`, `ansys.tools.visualization_interface.backends.pyvista.widgets.mesh_object_plot`
41 63
`ansys.tools.visualization_interface.backends.pyvista.widgets.utils`, `ansys.tools.visualization_interface.backends.pyvista.widgets.utils`
45 66
`ansys.tools.visualization_interface.backends.pyvista.widgets.utils.clip_plane`, `ansys.tools.visualization_interface.backends.pyvista.widgets.utils.clip_plane`
53 67
`ansys.tools.visualization_interface.backends.pyvista.widgets.utils.color`, `ansys.tools.visualization_interface.backends.pyvista.widgets.utils.color`
58 68
`ansys.tools.visualization_interface.backends.pyvista.widgets.trame_remote`, `ansys.tools.visualization_interface.backends.pyvista.widgets.trame_remote`
60 60
`ansys.tools.visualization_interface.backends.pyvista.widgets.trame_service`, `ansys.tools.visualization_interface.backends.pyvista.widgets.trame_service`
60 60
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets`
25 25
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.button`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.button`
26 26
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.dark_mode`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.dark_mode`
27 27
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.displace_arrows`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.displace_arrows`
28 28
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.hide_buttons`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.hide_buttons`
31 31
`ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.measure`, `ansys.tools.visualization_interface.backends.pyvista.widgets.widgets.measure`
32 32

Symbols

`__call__()` (in module *SingletonType*), 69
`__version__` (in module *visualization_interface*), 72

A

`actor` (in module *EdgePlot*), 63
`actor` (in module *MeshObjectPlot*), 65
`add_button()` (in module *ButtonManager*), 17
`add_child()` (in module *MeshObjectPlot*), 65
`add_file_handler()` (in module *VizLogger*), 70
`add_measurement_toggle_button()` (in module *ButtonManager*), 19
`add_mesh_name()` (in module *DashDropdownManager*), 21
`add_widget()` (in module *PyVistaBackendInterface*), 48
`ansys.tools.visualization_interface`
 module, 15
`ansys.tools.visualization_interface.backends`
 module, 15
`ansys.tools.visualization_interface.backends.plotly`
 module, 16
`ansys.tools.visualization_interface.backends.plotly_dash`
 module, 21
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 (built-in class), 22
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 module, 23
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 (built-in class), 23
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 module, 16
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 module, 16
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 (built-in class), 16
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 module, 20
`ansys.tools.visualization_interface.backends.plotly_dash.PlotlyDashBackend`
 (built-in class), 20
`ansys.tools.visualization_interface.backends.pyvista`
 module, 25
`ansys.tools.visualization_interface.backends.pyvista.picker`
 module, 41
`ansys.tools.visualization_interface.backends.pyvista.picker`
 (built-in class), 42
`ansys.tools.visualization_interface.backends.pyvista.picker`
 (built-in class), 43
`ansys.tools.visualization_interface.backends.pyvista.pyvista`
 module, 45
`ansys.tools.visualization_interface.backends.pyvista.pyvista`
 (built-in class), 51
`ansys.tools.visualization_interface.backends.pyvista.pyvista`
 (built-in class), 45
`ansys.tools.visualization_interface.backends.pyvista.pyvista`
 module, 53
`ansys.tools.visualization_interface.backends.pyvista.pyvista`
 (built-in class), 53
`ansys.tools.visualization_interface.backends.pyvista.trame`
 module, 58
`ansys.tools.visualization_interface.backends.pyvista.trame`
 (built-in class), 59
`ansys.tools.visualization_interface.backends.pyvista.trame`
 module, 60
`ansys.tools.visualization_interface.backends.pyvista.trame`
 module, 60
`ansys.tools.visualization_interface.backends.pyvista.trame`
 (built-in class), 61
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 module, 25
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 module, 26
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 (built-in class), 26
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 module, 27
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 (built-in class), 27
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 module, 28
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 (built-in class), 30
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 (built-in class), 28
`ansys.tools.visualization_interface.backends.pyvista.widgets`
 (built-in class), 28

direction (in module *ViewButton*), 39
 disable_center_focus() (in module *PyVistaBackendInterface*), 49
 disable_hover() (in module *PyVistaBackendInterface*), 49
 disable_picking() (in module *PyVistaBackendInterface*), 49
 DOCUMENTATION_BUILD (in module *visualization_interface*), 72
 dropdown_manager (in module *PlotlyDashBackend*), 22

E

EDGE (in module *Color*), 68
 edge_object (in module *EdgePlot*), 63
 edges (in module *MeshObjectPlot*), 65
 enable_hover() (in module *PyVistaBackendInterface*), 49
 enable_output() (in module *VizLogger*), 69
 enable_picking() (in module *PyVistaBackendInterface*), 49
 enable_set_focus_center() (in module *PyVistaBackendInterface*), 49
 enable_widgets() (in module *PyVistaBackendInterface*), 48

F

focus_point_selection() (in module *PyVistaBackendInterface*), 49

G

get_logger() (in module *VizLogger*), 69
 get_mesh_names() (in module *DashDropdownManager*), 21
 get_visibility_args_for_meshes() (in module *DashDropdownManager*), 21

H

hide_children() (in module *PyVistaInterface*), 57
 hover_callback() (in module *PyVistaBackendInterface*), 49
 hover_select_object() (in module *AbstractPicker*), 43
 hover_select_object() (in module *Picker*), 44
 hover_unselect_object() (in module *AbstractPicker*), 43
 hover_unselect_object() (in module *Picker*), 44

I

ISOMETRIC (in module *ViewDirection*), 40

L

layout (in module *PlotlyBackend*), 24
 logger (in module *logger*), 70

M

mesh (in module *MeshObjectPlot*), 65
 mesh_type (in module *MeshObjectPlot*), 65
 module
 ansys.tools.visualization_interface, 15
 ansys.tools.visualization_interface.backends, 15
 ansys.tools.visualization_interface.backends.plotly, 16
 ansys.tools.visualization_interface.backends.plotly.pl, 21
 ansys.tools.visualization_interface.backends.plotly.pl, 23
 ansys.tools.visualization_interface.backends.plotly.wi, 16
 ansys.tools.visualization_interface.backends.plotly.wi, 16
 ansys.tools.visualization_interface.backends.plotly.wi, 20
 ansys.tools.visualization_interface.backends.pyvista, 25
 ansys.tools.visualization_interface.backends.pyvista.p, 41
 ansys.tools.visualization_interface.backends.pyvista.p, 45
 ansys.tools.visualization_interface.backends.pyvista.p, 53
 ansys.tools.visualization_interface.backends.pyvista.t, 58
 ansys.tools.visualization_interface.backends.pyvista.t, 60
 ansys.tools.visualization_interface.backends.pyvista.t, 60
 ansys.tools.visualization_interface.backends.pyvista.w, 25
 ansys.tools.visualization_interface.backends.pyvista.w, 26
 ansys.tools.visualization_interface.backends.pyvista.w, 27
 ansys.tools.visualization_interface.backends.pyvista.w, 28
 ansys.tools.visualization_interface.backends.pyvista.w, 31
 ansys.tools.visualization_interface.backends.pyvista.w, 32
 ansys.tools.visualization_interface.backends.pyvista.w, 33
 ansys.tools.visualization_interface.backends.pyvista.w, 34
 ansys.tools.visualization_interface.backends.pyvista.w, 36
 ansys.tools.visualization_interface.backends.pyvista.w, 37
 ansys.tools.visualization_interface.backends.pyvista.w

38
 ansys.tools.visualization_interface.backends.pyvista_widgets_low_level_backend, 52
 40
 ansys.tools.visualization_interface.plotter, 71
 70
 ansys.tools.visualization_interface.types, 56
 62
 ansys.tools.visualization_interface.types.plotter_helper (in module MeasureWidget), 33
 62
 ansys.tools.visualization_interface.types.plotter_helper (in module MeshSliderWidget), 34
 ansys.tools.visualization_interface.types.mesh_object_plot (in module PickRotCenterButton), 35
 63
 pv_interface (in module PyVistaBackendInterface), 48
 ansys.tools.visualization_interface.utils, 61
 66
 ansys.tools.visualization_interface.utils.clip_plane (in module TrameService), 61
 66
 ansys.tools.visualization_interface.utils.color, 61
 67
 ansys.tools.visualization_interface.utils.logger (in module PyVistaInterface), 55
 68
 ansys.tools.visualization_interface.utils.send_mesh() (in module trame_remote), 60
 ansys.tools.visualization_interface.utils.send_pl() (in module trame_remote), 60
 ansys.tools.visualization_interface.utils.server (in module TrameVisualizer), 59
 ansys.tools.visualization_interface.utils.set_add_mesh_defaults() (in module PyVistaInterface), 58
 ansys.tools.visualization_interface.utils.set_level() (in module VizLogger), 69
 ansys.tools.visualization_interface.utils.set_scene() (in module TrameService), 61
 ansys.tools.visualization_interface.utils.set_scene() (in module TrameVisualizer), 59
 ansys.tools.visualization_interface.utils.show() (in module PlotlyBackend), 24
 ansys.tools.visualization_interface.utils.show() (in module PlotlyDashBackend), 23
 ansys.tools.visualization_interface.utils.show() (in module Plotter), 71
 ansys.tools.visualization_interface.utils.show() (in module PyVistaBackendInterface), 49
 ansys.tools.visualization_interface.utils.show() (in module PyVistaInterface), 58
 ansys.tools.visualization_interface.utils.show() (in module TrameVisualizer), 59
 ansys.tools.visualization_interface.utils.show_children() (in module PyVistaInterface), 57
 ansys.tools.visualization_interface.utils.show_hide_bbox_dict() (in module ButtonManager), 17
 ansys.tools.visualization_interface.utils.show_plotter() (in module PyVistaBackendInterface), 50
 ansys.tools.visualization_interface.utils.testing_mode (in module visualization_interface), 72
 ansys.tools.visualization_interface.utils.update() (in module Button), 27
 ansys.tools.visualization_interface.utils.update() (in module DarkModeButton), 28
 ansys.tools.visualization_interface.utils.update() (in module HideButton), 31
 ansys.tools.visualization_interface.utils.update() (in module MeasureWidget), 33
 ansys.tools.visualization_interface.utils.update() (in module MeshSliderWidget), 34
 ansys.tools.visualization_interface.utils.update() (in module PickRotCenterButton), 35
 ansys.tools.visualization_interface.utils.update() (in module PlotterWidget), 41
 ansys.tools.visualization_interface.utils.update() (in module Ruler), 37
 ansys.tools.visualization_interface.utils.update() (in module ScreenshotButton), 38
 ansys.tools.visualization_interface.utils.update_layout() (in module ButtonManager), 18
 ansys.tools.visualization_interface.utils.use_html_backend (in module visualization_interface), 72

N

name (in module EdgePlot), 63
 name (in module MeshObjectPlot), 65
 normal (in module ClipPlane), 67

O

object_to_actors_map (in module PyVistaInterface), 55
 origin (in module ClipPlane), 67

P

parent (in module EdgePlot), 63
 parent (in module MeshObjectPlot), 64
 pick_select_object() (in module AbstractPicker), 42
 pick_select_object() (in module Picker), 44
 pick_unselect_object() (in module AbstractPicker), 42
 pick_unselect_object() (in module Picker), 44
 PICKED (in module Color), 68
 picked_dict (in module AbstractPicker), 42
 picked_dict (in module Picker), 44
 PICKED_EDGE (in module Color), 68
 picked_operation() (in module PyVistaBackendInterface), 51
 picker_callback() (in module PyVistaBackendInterface), 49
 plot() (in module PlotlyBackend), 24
 plot() (in module PlotlyDashBackend), 22
 plot() (in module Plotter), 71
 plot() (in module PyVistaBackend), 52
 plot() (in module PyVistaBackendInterface), 50
 plot() (in module PyVistaInterface), 57
 plot_edges() (in module PyVistaInterface), 56
 plot_iter() (in module PlotlyBackend), 24
 plot_iter() (in module Plotter), 71
 plot_iter() (in module PyVistaBackendInterface), 50
 plot_iter() (in module PyVistaInterface), 57
 plot_meshobject() (in module PyVistaInterface), 56
 plotter (in module PlotterWidget), 41
 plotter (in module TrameVisualizer), 59
 plotter_helper (in module MeasureWidget), 33
 plotter_helper (in module MeshSliderWidget), 34
 mesh_object_plot (in module PickRotCenterButton), 35
 pv_interface (in module PyVistaBackendInterface), 48

R

S

scene (in module PyVistaBackendInterface), 48
 logger (in module PyVistaInterface), 55
 send_mesh() (in module trame_remote), 60
 send_pl() (in module trame_remote), 60
 server (in module TrameVisualizer), 59
 set_add_mesh_defaults() (in module PyVistaInterface), 58
 set_level() (in module VizLogger), 69
 set_scene() (in module TrameService), 61
 set_scene() (in module TrameVisualizer), 59
 show() (in module PlotlyBackend), 24
 show() (in module PlotlyDashBackend), 23
 show() (in module Plotter), 71
 show() (in module PyVistaBackendInterface), 49
 show() (in module PyVistaInterface), 58
 show() (in module TrameVisualizer), 59
 show_children() (in module PyVistaInterface), 57
 show_hide_bbox_dict() (in module ButtonManager), 17
 show_plotter() (in module PyVistaBackendInterface), 50

T

TESTING_MODE (in module visualization_interface), 72

U

update() (in module Button), 27
 update() (in module DarkModeButton), 28
 update() (in module HideButton), 31
 update() (in module MeasureWidget), 33
 update() (in module MeshSliderWidget), 34
 update() (in module PickRotCenterButton), 35
 update() (in module PlotterWidget), 41
 update() (in module Ruler), 37
 update() (in module ScreenshotButton), 38
 update_layout() (in module ButtonManager), 18
 USE_HTML_BACKEND (in module visualization_interface), 72

USE_FRAME (in module *visualization_interface*), [72](#)

V

`view_xy()` (in module *PyVistaInterface*), [56](#)

`view_xz()` (in module *PyVistaInterface*), [56](#)

`view_yx()` (in module *PyVistaInterface*), [56](#)

`view_yz()` (in module *PyVistaInterface*), [56](#)

`view_zx()` (in module *PyVistaInterface*), [56](#)

`view_zy()` (in module *PyVistaInterface*), [56](#)

X

XDOWN (in module *CameraPanDirection*), [30](#)

XUP (in module *CameraPanDirection*), [30](#)

XYMINUS (in module *ViewDirection*), [40](#)

XYPLUS (in module *ViewDirection*), [40](#)

XZMINUS (in module *ViewDirection*), [40](#)

XZPLUS (in module *ViewDirection*), [40](#)

Y

YDOWN (in module *CameraPanDirection*), [30](#)

YUP (in module *CameraPanDirection*), [30](#)

YZMINUS (in module *ViewDirection*), [40](#)

YZPLUS (in module *ViewDirection*), [40](#)

Z

ZDOWN (in module *CameraPanDirection*), [30](#)

ZUP (in module *CameraPanDirection*), [30](#)