View

Contents

class openstaadpy.os_analytical.osview.OSView [source]

Bases: object

CloseActiveWindow() [source]

Closes the active graphic or table window, however there must be at least one graphic window remaining open. Note that window at position 1 can not be closed.

Returns:

Returns True if Window closed. Returns FALSE if Unsuccessful.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> output = staad_obj.View.CloseActiveWindow()
```

CopyPicture() [source]

Copy active view to clipbord and gives size of image in it's referance variables. It does not copy table views but works with Structure View, Graphs, 3d Rendered view.

Returns:

Returns a tuple containing xDim size of image in x direction (Length in Pixel) and yDim size of image in y direction (width in Pixel) respectively.

Return type:

Tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.CopyPicture()
```

CreateNewViewForSelections() [source]

Creates a new view in new window for the selected objects displayed in the active window.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.CreateNewViewForSelections()
```

CreateNewViewForSelectionsEx(*windowOptions: int*) [source]

Creates a new view for the selected objects displayed in the active window based on specified options.

Parameters:

windowOptions (*int*) – 0 = Creates a new window for the view, 1 = Display the view in the active window (type - Long).

Returns:

Returns True Creation of new view is successful. Returns False Creation of new view is unsuccessful.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.CreateNewViewForSelectionsEx(1)
```

DetachView() [source]

Remove a view from the collection of saved views. The view to be removed must be open and the active window. Once the view is detached, the active window becomes <untitled>. To set the active window see the function OSViewUI::SetActiveWindow. Note that the active window view cannot be Whole Structure or untitled.

Returns:

Returns 1 if View successfully detached. Returns 0 if Unsuccessful Returns -1 if Generic Error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> status = staad_obj.View.DetachView()
```

ExportView(*FileLocation: str, FileName: str, FileFormat: int, Overwrite: bool*) [source]

Used for exporting the information displayed in the active view window into a standard, graphical image format (e.g., Bitmap, JPEG, TIFF et.).

Parameters:

- **FileLocation** (*str*) – Location of the saved view file (Folder need to be present otherwise it will return -100) .
- **FileName** (*str*) – Name of the saved view file.

- **FileFormat** (*int*) – 0 = bmp, 1 = jpg, 2 = tga, 3 = tif - Create the view in the specific format.
- **Overwrite** (*bool*) –

  Boolean for provide option to overwrite an existing file.

  - True - Allow Overwrite
  - False - No overwrite

Returns:

Returns 1 if Export view is successful. Returns -1 if Generic Error. Returns -100 if Invalid Argument. Returns -1003 if File already exist.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ExportView(r"<folderPath>", "<fileName>", 1, True)
```

GetApplicationDesktopSize() [source]

Get the size of the application desktop.

Returns:

(width, height) of the application desktop.

Return type:

tuple of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> width, height = staad_obj.View.GetApplicationDesktopSize()
```

GetBeamsInView(*nBeamList: list*) [source]

Get Beams In View

Parameters:

nBeamList (*nBeamList*) – Collection of beam

Returns:

Returns number of beams present in view.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.GetBeamsInView([1, 2, 4])
```

GetInterfaceMode() [source]

This function returns the current visual mode in the STAAD.Pro environment.

Returns:

Returns 0 if Pre-processor or modeling mode. Returns 1 if Post-processing mode. Returns 2 if Interactive design mode for STAAD.etc interoperability. Returns 4 if Piping mode. Returns 5 if BEAVA (i.e., Bridge Deck) mode.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.GetInterfaceMode()
```

GetNoOfBeamsInView() [source]

Get No Of Beams In View

Returns:

Returns number of beams present in view.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.GetNoOfBeamsInView()
```

GetScaleCount() [source]

Returns the count of scales that are used in STAAD.Pro which can be read and set using the functions GetScaleValues and SetScaleValues.

Returns:

Returns Positive_Value Count of scales. Returns 0/Negative_Value Unsuccessful.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> scaleCount = staad_obj.View.GetScaleCount()
```

GetScaleValueByType(*scaleTypeId: int*) [source]

Obtain the value of the scale that is used to display a specified load or result diagram as defined in the Diagrams>Scales dialog depending upon the scale ID passed.

Parameters:

scaleTypeId (*int*) – The index of the required load or result type

Returns:

Returns value of scale type listed, in Base Units

Return type:

float

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> value = staad_obj.View.GetScaleValueByType(1)
>>> print(value)
```

GetScaleValues() [source]

Obtain the current set of scales used for displaying loads and results shown in the Diagrams > Scales dialog.

Returns:

Returns list of float type and size same as number of scale types. API gets all the values in this list. Scale type and ID in array will be as following:

ID	Type	Scale Items	Unit per length
0	Loads	Point Force	Force
1	Loads	Dist. Force	Force/length
2	Loads	Point Moment	Force*length
3	Loads	Dist. Moment	Force*length/length
4	Loads	Pressure	Force/length^2
5	Results	Bending Y	Force*length
6	Results	Bending Z	Force*length
7	Results	Shear Y	Force
8	Results	Shear Z	Force
9	Results	Axial	Force
10	Results	Torsion	Force*length
11	Results	Displacement	Length
12	Results	Beam Stress	Force/length^2
13	Results	Mode Shape	(none)

Return type:

list of float

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> list = staad_obj.View.GetScaleValues()
>>> print(list)
```

GetWindowCount()

[source]

Get the number of windows currently open. This includes both graphic windows and tables.

Returns:

Returns Positive_Number The count of open Window. Returns -1 if Error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.View.GetWindowCount()
```

GetWindowTitle(*id: int*)

[source]

Returns the Title of the Window.

Parameters:

id (*int*) – The index of the required Window (Type: Long). Note that IDs start from 1. Window IDs depend on the creation time; that is the last created window will have the last ID. Windows which are listed under in View tab > Windows Dropdown are supported in this API.

Returns:

Returns the Window string title. Returns Empty_String Window id not found.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> title = staad_obj.View.GetWindowTitle(1)
```

HideAllMembers() [source]

Hide all members in the STAAD view.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideAllMembers()
```

HideEntity(*nEntity: int*) [source]

Hides the specified entity, which may be a Beam, Plate, Solid, or Surface.

Parameters:

nEntity (*int*) – Variable that holds an entity (i.e., Member, Plates etc.) number to be hidden.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideEntity(5)
```

HideMember(*IDMember*) [source]

Hide a specific member in the STAAD view.

Parameters:

IDMember (*int*) – Member number to hide.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideMember(1)
```

HideMembers(*NMembers, NaMemberNos*) [source]

Hide specific members in the STAAD view.

Parameters:

- **NMembers** (*int*) – Number of members to hide.
- **NaMemberNos** (*list of int*) – List of member numbers to hide.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideMembers(2, [1, 2])
```

HidePlate(*nPlate: int*) [source]

Hide the specified plate.

Parameters:

nPlate (*int*) – Variable that holds plate number to be hidden.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HidePlate(5)
```

HideSolid(*nSolid: int*) [source]

Hide the specified solid.

Parameters:

nSolid (*int*) – Variable that holds solid number to be hidden.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideSolid(5)
```

HideSurface(*nSurface: int*) [source]

Hide the specified surface.

Parameters:

nSurface (*int*) – Variable that holds surface number to be hidden.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.HideSurface(5)
```

OpenView(*viewName: str, windowOptions: bool*) [source]

Open a previously saved view in either the active window or create a new window which becomes the active window.

Parameters:

- **viewName** (*str*) – New name of the saved view
- **windowOptions** (*bool*) – False = Creates a new window for the view which becomes the active window, True / 1 = Display the view in the current active window

Returns:

Returns 1 if View Successfully opened. Returns 0 if Unsuccessful Returns 2 if View name does not exist. Returns -1 if Generic Error Returns -100 if Invalid Argument

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> status = staad_obj.View.OpenView("view1", True)
>>> print(status)
```

RefreshView() [source]

Refresh the STAAD view window.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.RefreshView()
```

RenameView(*viewName: str*) [source]

Renames a saved view. The view should be open and be the active window.

Parameters:

viewName (*str*) – New name of the saved view

Returns:

Returns 1 if Rename view is successful. Returns 0 if Unsuccessful Returns 2 if View name already used. Returns -1 if Generic Error Returns -100 if Invalid Argument

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> status = staad_obj.View.RenameView("view1")
```

RotateDown(*dDegrees: float*) [source]

Rotates the structure through Degrees about the Global X-Axis.

Parameters:

dDegrees (*float*) – Variable providing the degree of rotation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.RotateDown(30)
```

RotateLeft(*dDegrees: float*) [source]

Rotates the structure through Degrees about the Global Y-Axis.

Parameters:

dDegrees (*float*) – Variable providing the degree of rotation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.RotateLeft(30)
```

RotateRight(*dDegrees: float*) [source]

Rotates the structure through Degrees about the Global Y-Axis.

Parameters:

dDegrees (*float*) – Variable providing the degree of rotation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.RotateRight(30)
```

RotateUp(*dDegrees: float*) [source]

Rotates the structure through Degrees about the Global X-Axis.

Parameters:

dDegrees (*float*) – Variable providing the degree of rotation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.RotateUp(30)
```

SaveView(*viewName: str, overWrite: bool*) [source]

Save the active graphic view to the collection of saved views which can be opened using the function OpenView.

Parameters:

- **viewName** (*str*) – New name for the view
- **overWrite** (*bool*) – Option to overwrite if the given viewName already exists. False = Do not overwrite., TRUE = Overwrite viewName if it exists.

Returns:

Returns 1 if Save view is successful. Returns 0 if Unsuccessful Returns 2 if View name already exist and overWrite is false. Returns -1 if Generic Error Returns -100 if Invalid Argument

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> status = staad_obj.View.SaveView("view1", True)
>>> print(status)
```

SelectByItemList(*entityType: int, nItems: int, itemList: list*)

Select entities as specified. [source]

Parameters:

- **entityType** (*int*) –

  Variable that holds entity type. Values may be as follows:

ID	Entity Type
1	Node
2	Beam
3	Plate
4	Solid
5	Surface

- **nItems** (*int*) – Variable that holds total number of entities needs to be selected.
- **itemList** (*list of int*) – List holds the entity nos, which need to be selected.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectByItemList(1, 2, [1, 2])
```

SelectByMissingAttribute(*attributeCode: int*) [source]

Select entity list for which specified entity is missing.

Parameters:

attributeCode (*int*) –

Variable that holds attribute type. Values may be as follows:

ID	Entity Type
1	Missing Property
2	Missing Modulus of Elasticity
3	Missing Density of Material
4	Missing Alpha (Coefficient of Thermal Expansion)
5	Missing Poisson Ratio

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectByMissingAttribute(5)
```

SelectEntitiesConnectedToMember(*entityType: int, memberNo: int*)

Select entities as specified in type and connected with the specified Member. [source]

Parameters:

- **entityType** (*int*) –

  Variable that holds entity type. Values may be as follows:

ID	Entity Type
0	Geometry
1	Beam
2	Plate
3	Solid

- **memberNo** (*int*) – Variable that holds Member numbers with which connected entities needs to be selected.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectEntitiesConnectedToMember(1, 2)
```

SelectEntitiesConnectedToNode(*entityType: int, nodeNo: int*)

Select entities as specified in type and connected with the specified node. [source]

Parameters:

- **entityType** (*int*) –

  Variable that holds entity type. Values may be as follows:

ID	Entity Type
0	Geometry
1	Beam
2	Plate
3	Solid

- **nodeNo** (*int*) – Variable that holds node numbers with which connected entities needs to be selected.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectEntitiesConnectedToNode(0, 1)
```

SelectEntitiesConnectedToPlate(*entityType: int, plateNo: int*)

Select entities as specified in type and connected with the specified Plate. [source]

Parameters:

- **entityType** (*int*) –

  Variable that holds entity type. Values may be as follows:

ID	Entity Type
0	Geometry
1	Beam
2	Plate
3	Solid

- **plateNo** (*int*) – Variable that holds Plate numbers with which connected entities needs to be selected.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectEntitiesConnectedToPlate(2, 3)
```

SelectEntitiesConnectedToSolid(*entityType: int, solidNo: int*)

Select entities as specified in type and connected with the specified Solid. [source]

Parameters:

- **entityType** (*int*) –

  Variable that holds entity type. Values may be as follows:

ID	Entity Type
0	Geometry
1	Beam
2	Plate
3	Solid

- **solidNo** (*int*) – Variable that holds Solid numbers with which connected entities needs to be selected.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectEntitiesConnectedToSolid(3, 4)
```

SelectGroup(*bstrGroup: str*) [source]

Select the relevant entities of the specified group.

Parameters:

bstrGroup (*str*) – A string variable that holds the group name.

Returns:

Returns True if successful Returns False if unsuccessful

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectGroup("grp_name")
```

SelectInverse(*entityType: int*) [source]

Inverse geometry selection for the specified entity.

Parameters:

entityType (*int*) –

Variable that holds entity type. Values may be as follows:

ID	Entity Type
1	Node
2	Beam
3	Plate
4	Solid
5	Surface

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectInverse(1)
```

SelectMembersParallelTo(*bstrAxis: str*) [source]

Select members parallel to the specified axis.

Parameters:

bstrAxis (*str*) –

Variable that holds the Axis ID. It may have three values:

ID	Axis
X	X-Axis
Y	Y-Axis
Z	Z-Axis

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SelectMembersParallelTo(5)
```

SetActiveWindow(*id: int*) [source]

Set a given window (active graphic or table window) with the provided id as the active window. The indexing starts from 1.

Parameters:

id (*int*) – The id of the window to be made the active window.

Returns:

Returns True if successful. Returns FALSE if unsuccessful.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetActiveWindow(2)
```

SetBeamAnnotationMode(*Type: int, DWFlags: int, RefreshFlag: bool*) [source]

This function sets the current page mode in the STAAD.Pro environment.

Parameters:

- **Type** (*int*) –

  Variable controlling the annotation type. It may be one of the following values:

ID	Annotation Type
0	Axial Diagram
1	Torsion Diagram
2	Moment Diagram
3	Shear Diagram
4	Stress Diagram
5	Displacement Diagram

- **DWFlags** (*int*) –

  Variable controlling what values are to be shown for the annotationType. It may be one of the following values:

ID	Values
1	End Values
2	Max Absolute Values
3	Mid-span Values

- **RefreshFlag** (*int*) – Boolean variable (True or False). If True, STAAD.Pro viewing windows refresh with the current annotation mode.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetBeamAnnotationMode(2, 1, True)
```

SetDesignResults(*utilization: int, color: bool, showValues: bool*)

Sets Design Results to active view, this function replicates the setting of Design Results in the Diagrams>Design Results dialog. [source]

Parameters:

- **Id** (*int*) – Value of type Long. (0 = None, 1 = Actual Ratio, 2 = Normalised Ratio.)
- **color** (*bool*) – Value of type Boolean. (False/0 = Basic Colored, True/1 = Detailed Colored.)
- **showValues** (*bool*) – Value of type Boolean. (False/0 = Do Not Show Values, True/1 = Show Values.)

Returns:

Returns 1 if Set Design Results is successful. Returns 0 if Unsuccessful. Returns -1 if Generic Error. Returns -2 if Design Results not Loaded. Returns -100 if Invalid Argument.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetDesignResults(1, True, True)
```

SetDiagramMode(*which: int, showFlag: bool, refreshFlag: bool*)

Sets the label on the structure diagram on or off. [source]

Parameters:

- **which** (*int*) –

Variable identifying the diagram type. It may be one of the following values

ID	Diagram Type
0	Load
1	Displacement
2	MY
3	MZ
4	FY
5	FZ
6	AX
7	TR
8	Structure
9	Full Section
10	Section Outline
11	Stress
12	Shrink
13	Perspective
14	Hide Structure
15	Fill Plates & Solids
16	Hide Plates & Solids
18	Hide Piping
19	Sort Geometry
20	Sort Nodes
21	Plate Stress

ID	Diagram Type
22	Solid Stress
23	Mode Shape
24	Stress Animation
25	Plate reinforcement
26	Deck Influence Diagram*
27	Deck Carriageways*
28	Deck Triangulation*
29	Deck Loads*
30	Deck Vehicles*
	(*) Requires the STAAD.beava component

- **showFlag** (*bool*) – Variable to set label mode on (True) or off (False).
- **refreshFlag** (*bool*) – Variable (True or False). If True, STAAD.Pro viewing windows refresh.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetDiagramMode(1, True, True)
```

SetInterfaceMode(*interfaceMode: int*) [source]

This function sets the current visual mode in the STAAD.Pro environment.

Parameters:

interfaceMode (*int*) –

Variable to set the current visual mode in STAAD.Pro environment.

Followings are the valid values for mode:

ID	Mode Type
0	Pre-processor or modeling mode
1	Physical modeling mode
2	Building planner mode
3	Piping mode
5	Post Processing mode
6	FoundationDesign mode
7	ConnectionDesign mode
9	AdvancedConcreteDesign mode
10	AdvancedSlabDesign mode
11	Earthquake mode
12	SteelAutoDrafter mode
13	ChineseSteelDesign mode

Returns:

Returns 0 if Pre-processor or modeling mode. Returns 1 if Post-processing mode. Returns 2 if Interactive design mode for STAAD.etc interoperability. Returns 4 if Piping mode. Returns 5 if BEAVA (i.e., Bridge Deck) mode.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetInterfaceMode(1)
```

`SetLabel(which: int, showFlag: bool)` [source]

Sets the label on the structure diagram on or off.

Parameters:

- **which** (*int*) –

  Variable identifying the diagram type. It may be one of the following values:

ID	Label Type
0	Node number label
1	Member number label
2	Member property reference label
3	Material property reference label
4	Support label
5	Member release label
6	Member orientation label
7	Member section label
8	Load value label
9	Axes label
10	Node position label
11	Member specification label
12	Member ends
13	Plate element number label
14	Plate element orientation label
15	Solid element number label
16	Dimension label
17	Floor load label
18	Floor load distribution diagram label
19	Wind load label
20	Wind load influence area diagram label

ID	Label Type
21	Diagram Info

- **showFlag** (*bool*) – Variable to set label mode on (True) or off (False).

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetLabel(20, True)
```

SetModeSectionPage(*interfaceMode: int, sectionNumber: int, pageNumber: int*) [source]

This function sets the current page mode in the STAAD.Pro environment.

Parameters:

- **interfaceMode** (*int*) –

  Variable to set the current visual mode in STAAD.Pro environment. Followings are the valid values for mode:

ID	Interface Mode
0	Pre-processor or modeling mode
1	Post-processing mode
2	Interactive design mode for STAAD.etc interoperability
4	Piping mode
5	BEAVA (i.e., Bridge Deck) mode

- **sectionNumber** (*int*) –

  Variable to set the current main page (the tabs on the left-hand side of the screen) in the STAAD.Pro environment. The following are valid values for section:

ID	Main Page
1	Setup page
2	Geometry page
3	General page
5	Node Results page
6	Beam Result page
7	Plate Results page
8	Solid Results page

- **pageNumber** (*int*) –

  Variable to set the current sub page (within a particular main page - the tabs on the left-hand side of the screen) in the STAAD.Pro environment. The following are valid values for modeSubPage:

ID	Page Number
0	Job Info page
1	Beam page
4	Plate page
5	Solid page
6	Property page
7	Constant page
8	Material page
9	Support page
10	Member Specifications page
11	Load page
17	Reaction page
18	Displacement page
19	Failure page
20	Forces page
21	Beam Stress page
22	Plate Stress page
23	Solid Stress page

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetModeSectionPage(1,6,20)
```

SetNodeAnnotationMode(*dFlag: bool, refreshFlag: bool*) [source]

Sets the node displacement annotation mode. This function works only in the post-processing mode of STAAD.Pro.

Parameters:

- **dFlag** (*bool*) –

  Variable controlling the annotation type. It may be one of the following values:

ID	Annotation Type
1	X Displacement
2	Y Displacement
3	Z Displacement
4	Resultant Displacement

- **refreshFlag** (*bool*) – Variable (True or False). If True, STAAD.Pro viewing windows refresh with the current annotation mode.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetNodeAnnotationMode(1, True)
```

SetReactionAnnotationMode(*dFlag: bool, refreshFlag: bool*)

Sets the node displacement annotation mode. This function works only in the post-processing mode of STAAD.Pro. [source]

Parameters:

- **dFlag** (*bool*) –

  Variable controlling the annotation type. It may be one of the following values:

ID	Annotation Type
1	X Reaction
2	Y Reaction
3	Z Reaction
4	X Rotation
5	Y Rotation
6	Z Rotation
7	Reaction Value Only

- **refreshFlag** (*bool*) – Variable (True or False). If True, STAAD.Pro viewing windows refresh with the current annotation mode.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetReactionAnnotationMode(1, True)
```

SetScaleValueByType(*scaleTypeId: int, value: float*) [source]

Set the scale used for displaying a chosen load or result diagram as shown in the Diagrams>Scales dialog.

Parameters:

- **scaleTypeId** (*int*) – The index of the required load or result type to be set.
- **value** (*float*) – Value of scale type to be used.

Returns:

Returns 1/TRUE Value was successfully updated. Returns 0/FALSE Value could not be updated.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> retValue = staad_obj.View.SetScaleValueByType(1, 1.2)
```

SetScaleValues(*ScalesList: list*) [source]

Set the scales used for displaying loads and results as shown in the Diagrams>Scales dialog.

Parameters:

ScalesList (*list of float*) – List of float type and size same as number of scale types. API sets the values in this list to scale types

Returns:

Returns 1 if Values were successfully updated. Returns 0 if Values could not be updated. Returns -1 if Error with defined array.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetScaleValues([1.0, 2.5, 3.2])
```

SetSectionView(*plane: int, minVal: float, maxVal: float*) [source]

Creates a section view of the structure.

Parameters:

- **plane** (*int*) –

  Variable identifying the section plane. It may be one of the following values:

ID	Values for plane
0	XY Plane
1	YZ Plane
2	XZ Plane

- **minVal** (*float*) – Minimum range of the cutting plane.
- **maxVal** (*float*) – Maximum range of the cutting plane.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetSectionView(1, 0.4, 0.6)
```

SetUnits(*uType: int, strUnit: str*) [source]

Set viewing unit for the active view.

Parameters:

- **uType** (*int*) –

  Variable that holds unit type. Values are as follows:

ID	Unit Type
0	Dimension
1	Displacement
2	SectionDimension
3	SectionArea
4	Inertia
5	Force
6	Moment
7	DistributedForce
8	DistributedMoment
9	Density
10	Acceleration
11	Spring
12	RotSpring
13	MaterialModulus
14	Stress
15	Alpha
16	Temperature
17	Mass
18	SectionModulus
19	RotationalDisplacement
20	SubgradeModulus

ID	Unit Type
-1	NoUnit

- **strUnit** (*str*) – Variable array that holds the unit for the specified type. Like “cm”, “kns”, “feet”, “kn/cm” etc.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetUnits(0, "cm")
```

SetWindowPosition(*xTop, yTop, xWindow, yWindow*) [source]

Set the position and size of the STAAD application window.

Parameters:

- **xTop** (*int*) – X coordinate of the top-left corner.
- **yTop** (*int*) – Y coordinate of the top-left corner.
- **xWindow** (*int*) – Width of the window.
- **yWindow** (*int*) – Height of the window.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SetWindowPosition(100, 100, 800, 600)
```

ShowAllMembers() [source]

Show all members in the STAAD view.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowAllMembers()
```

ShowBack() [source]

Set the view to the back orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowBack()
```

ShowBottom() [source]

Set the view to the bottom orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowBottom()
```

ShowFront() [source]

Set the view to the front orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowFront()
```

ShowIsometric() [source]

Set the view to isometric orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowIsometric()
```

ShowLeft() [source]

Set the view to the left orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowLeft()
```

ShowMember(*nMember: int*) [source]

Show the specified member.

Parameters:

nMember (*int*) – Variable that holds member number to be shown.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowMember(5)
```

ShowMembers(*NMembers, NaMemberNos*) [source]

Show specific members in the STAAD view.

Parameters:

- **NMembers** (*int*) – Number of members to show.
- **NaMemberNos** (*list of int*) – List of member numbers to show.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowMembers(2, [1, 2])
```

ShowPlan() [source]

Set the view to the plan (top) orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowPlan()
```

ShowRight() [source]

Set the view to the right orientation.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ShowRight()
```

SpinLeft(*Degrees*) [source]

Spin the view to the left by a specified number of degrees.

Parameters:

Degrees (*float or int*) – Number of degrees to spin left.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SpinLeft(15)
```

SpinRight(*Degrees*) [source]

Spin the view to the right by a specified number of degrees.

Parameters:

Degrees (*float or int*) – Number of degrees to spin right.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.SpinRight(15)
```

ZoomAll() [source]

Zoom to show all objects in the STAAD view.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ZoomAll()
```

ZoomExtentsMainView() [source]

Zoom to extents in the main STAAD view.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.View.ZoomExtentsMainView()
```

__init__(*staadObj*) [source]