```
#-------------------------------------------------------------------------------
# Copyright (c) Bentley Systems, Incorporated. All rights reserved.
# See COPYRIGHT.md in the repository root for full copyright notice
#-------------------------------------------------------------------------------
from .openStaadHelper import *
from .oserrors import *
from comtypes import automation
from comtypes import CoInitialize
```

[docs]

```
class OSGeometry:
    CoInitialize()
```

[docs]

```
    def __init__(self, staadObj):
        self._staad = staadObj
        self._geometry = self._staad.Geometry

        self._functions= [
            "CreateNode",
            "CreateBeam",
            "CreatePlate",
            "CreateSolid",
            "AddNode",
            "AddBeam",
            "AddPlate",
            "AddSolid",
            "AddMultipleNodes",
            "AddMultipleBeams",
            "AddMultiplePlates",
            "AddMultipleSolids",
            "DeleteNode",
            "DeleteBeam",
            "DeletePlate",
            "DeleteSolid",
            "SplitBeam",
            "SplitBeamInEqlParts",
            "GetLastNodeNo",
            "GetLastBeamNo",
            "GetLastPlateNo",
            "GetLastSolidNo",
            "GetNoOfSelectedNodes",
            "GetSelectedNodes",
            "GetNoOfSelectedBeams",
            "GetSelectedBeams",
            "GetNoOfSelectedPlates",
            "GetSelectedPlates",
            "GetNoOfSelectedSolids",
            "GetSelectedSolids",
            "GetNodeCoordinates",
            "GetNodeNumber",
            "GetNodeDistance",
            "GetBeamLength",
            "SelectMultipleNodes",
            "SelectMultipleBeams",
            "SelectMultiplePlates",
            "SelectMultipleSolids",
            "SelectNode",
            "SelectBeam",
            "SelectPlate",
            "SelectSolid",
            "GetNodeCount",
            "GetMemberCount",
            "GetPlateCount",
            "GetSolidCount",
            "GetNodeList",
            "GetBeamList",
            "GetPlateList",
            "GetSolidList",
            "GetNodeIncidence",
            "GetMemberIncidence",
            "GetPlateIncidence",
            "GetSolidIncidence",
            "CreateGroup",
            "ClearNodeSelection",
            "ClearMemberSelection",
            "ClearPlateSelection",
            "ClearSolidSelection",
            "SetNodeUniqueID",
            "SetMemberUniqueID",
            "SetPlateUniqueID",
            "SetSolidUniqueID",
            "SetNodeCoordinate",
            "DoTranslationalRepeat",
            "GetNodeUniqueID",
            "GetMemberUniqueID",
            "GetPlateUniqueID",
            "GetSolidUniqueID",
            "GetPlateNodeCount",
            "GetNoOfGeneratedQuadPanels",
            "GetGeneratedQuadPanelIncidences",
            "IsZUp",
            "IsBeam",
            "IsColumn",
            "GetNoOfBeamsConnectedAtNode",
            "GetBeamsConnectedAtNode",
            "RenumberBeam",
            "IsOrphanNode",
            "GetGroupCountAll",
            "GetGroupCount",
            "GetGroupNames",
            "GetGroupEntityCount",
            "GetGroupEntities",
            "CreateGroupEx",
            "DeleteGroup",
            "UpdateGroup",
            "DefineParametricSurface",
            "AddParametricSurfaceToModel",
            "CommitParametricSurfaceMesh",
            "RemoveParametricSurfaceMesh",
            "AddDensityPointToSurface",
            "AddDensityLineToSurface",
            "AddCircularRegionToSurface",
            "AddPolygonalRegionToSurface",
            "GetParametricSurfaceCount",
            "GetParametricSurfaceInfo",
            "GetParametricSurfaceMeshInfo",
            "GetParametricSurfaceMeshData",
            "SetParametricSurfaceUniqueID",
            "GetParametricSurfaceUniqueID",
            "GetAreaOfPlates",
            "CreateMultiplePlates",
            "SetParametricSurfaceSubType",
            "GetParametricSurfaceSubType",
            "SetCheckForIdenticalEntity",
            "CreateMultipleNodes",
            "CreateMultipleBeams",
            "GetParametricSurfaceInfoEx",
            "IntersectBeams",
            "MergeBeams",
            "MergeNodes",
            "GetCountOfBreakableBeamsAtSpecificNodes",
            "BreakBeamsAtSpecificNodes",
            "GetIntersectBeamsCount",
            "ClearPhysicalMemberSelection",
            "CreatePhysicalMember",
            "DeletePhysicalMember",
            "GetAnalyticalMemberCountForPhysicalMember",
            "GetAnalyticalMembersForPhysicalMember",
            "GetLastPhysicalMemberNo",
            "GetNoOfSelectedPhysicalMembers",
            "GetSelectedPhysicalMembers",
            "GetPhysicalMemberCount",
            "GetPhysicalMemberList",
            "GetPhysicalMemberUniqueID",
            "GetPMemberCount",
            "SelectMultiplePhysicalMembers",
            "SelectPhysicalMember",
            "SetPhysicalMemberUniqueID",
            "SetPID",
            "GetPID",
            "GetFlagForHiddenEntities",
            "GetMemberIncidence_CIS2",
            "GetNodeIncidence_CIS2",
            "GetPlateIncidence_CIS2",
            "GetSolidIncidence_CIS2",
            "SetCheckForIdenticalEntity",
            "SetFlagForHiddenEntities"
        ]

        for function_name in self._functions:
            self._geometry._FlagAsMethod(function_name)
```

```
    ## NODE FUNCTIONS
```

[docs]

```
    def GetLastNodeNo(self):
        """
        Get the last node number.

        Returns
        -------
        int
            The last node number.
            - 1 : General error.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> last_node = staad_obj.Geometry.GetLastNodeNo()
        >>> print(last_node)
        """
        result = self._geometry.GetLastNodeNo()
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def GetNodeCoordinates(self,node:int):
        """
        Get the coordinates of a node.

        Parameters
        ----------
        node : int
            Node number.

        Returns
        -------
        tuple of float
            (x, y, z) coordinates.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> coords = staad_obj.Geometry.GetNodeCoordinates(1)
        >>> print(coords)
        """
        safe_n1 = make_safe_array_double(1)
        x = make_variant_vt_ref(safe_n1,  automation.VT_R8)

        safe_n2 = make_safe_array_double(1)
        y = make_variant_vt_ref(safe_n2,  automation.VT_R8)
        safe_n3 = make_safe_array_double(1)
        z = make_variant_vt_ref(safe_n3,  automation.VT_R8)

        self._geometry.GetNodeCoordinates(node,x,y,z)

        return (x[0],y[0],z[0])
```

[docs]

```
    def GetNodeCount(self):
        """
        Get the total number of nodes.

        Returns
        -------
        int
            Number of nodes.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetNodeCount()
        >>> print(count)
        """
        return self._geometry.GetNodeCount()
```

[docs]

```
    def GetNodeDistance(self,nodeA, nodeB):
        """
        Get the distance between two nodes.

        Parameters
        ----------
        nodeA : int
            First node number.
        nodeB : int
            Second node number.

        Returns
        -------
        float
            Distance between the nodes.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> dist = staad_obj.Geometry.GetNodeDistance(1, 2)
        >>> print(dist)
        """
        result = self._geometry.GetNodeDistance(nodeA,nodeB)
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def GetNodeIncidence(self,node):
        """
        Get the incidence (coordinates) of a node.

        Parameters
        ----------
        node : int
            Node number.

        Returns
        -------
        tuple
            (x, y, z) Coordinates of the node.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> inc = staad_obj.Geometry.GetNodeIncidence(1)
        >>> print(inc)
        """
        x_vt = make_variant_vt()
        x = make_variant_vt_ref(x_vt,  automation.VT_R8)

        y_vt = make_variant_vt()
        y = make_variant_vt_ref(y_vt,  automation.VT_R8)

        z_vt = make_variant_vt()
        z = make_variant_vt_ref(z_vt,  automation.VT_R8)

        result = self._geometry.GetNodeIncidence(node,x,y,z)
        if result < 0:
            raise_os_error_if_error_code(result)

        return x[0],y[0],z[0]
```

[docs]

```
    def GetNodeList(self):
        """
        Get the list of all node numbers.

        Returns
        -------
        list
            List of node numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> nodes = staad_obj.Geometry.GetNodeList()
        >>> print(nodes)
        """
        n_nodes =  int(self._geometry.GetNodeCount())
        if n_nodes <= 0:
            return []
        safe_list = make_safe_array_long(n_nodes)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetNodeList(lista)

        return list(lista[0])
```

[docs]

```
    def GetNodeNumber(self,x_y_z_coordinates:tuple):
        """
        Get the node number from coordinates.

        Parameters
        ----------
        x_y_z_coordinates : tuple of float
            (x, y, z) coordinates.

        Returns
        -------
        int
            Node number.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> node_no = staad_obj.Geometry.GetNodeNumber((0.0, 0.0, 0.0))
        >>> print(node_no)
        """
        result = self._geometry.GetNodeNumber(x_y_z_coordinates[0],x_y_z_coordi
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def GetNoOfSelectedNodes(self):
        """
        Get the number of selected nodes.

        Returns
        -------
        int
            Number of selected nodes.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> no_of_selected_nodes = staad_obj.Geometry.GetNoOfSelectedNodes()
        >>> print(no_of_selected_nodes)
        """
        return self._geometry.GetNoOfSelectedNodes()
```

[docs]

```
    def GetSelectedNodes(self):
        """
        Get the list of selected node numbers.

        Returns
        -------
        list
            Selected node numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> selected = staad_obj.Geometry.GetSelectedNodes()
        >>> print(selected)
        """
        n_nodes = self.GetNoOfSelectedNodes()
        safe_list = make_safe_array_long(n_nodes)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetSelectedNodes(lista)

        return list(lista[0])



    ## BEAM FUNCTIONS
```

[docs]

```
    def GetBeamLength(self,beam:int):
        """
        Get the length of a beam.

        Parameters
        ----------
        beam : int
            Beam number.

        Returns
        -------
        float
            Length of the beam.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> length = staad_obj.Geometry.GetBeamLength(1)
        >>> print(length)
        """
        result = self._geometry.GetBeamLength(beam)
        return result



    def GetMemberCount(self):
        """
        Get the number of beams.

        Returns
        -------
        int
            Number of beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetMemberCount()
        >>> print(count)
        """
        return self._geometry.GetMemberCount()


                                                                          [docs]
    def GetBeamList(self):
        """
        Get the list of all beam numbers.

        Returns
        -------
        list of int
            List of beam numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> beams = staad_obj.Geometry.GetBeamList()
        >>> print(beams)
        """
        beams = self._geometry.GetMemberCount()
        if beams <= 0:
            return []

        safe_list = make_safe_array_long(beams)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetBeamList(lista)

        return list(lista[0])
```

[docs]

```
    def GetLastBeamNo(self):
        """
        Get the last beam ID.

        Returns
        -------
        int
            Last beam number.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> last_beam = staad_obj.Geometry.GetLastBeamNo()
        >>> print(last_beam)
        """
        result = self._geometry.GetLastBeamNo()
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def GetMemberCount(self):
        """
        Get number of beam.

        Returns
        -------
        int
            Number of beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> beam_count = staad_obj.Geometry.GetMemberCount()
        >>> print(beam_count)
        """
        return self._geometry.GetMemberCount()
```

[docs]

```
    def GetMemberIncidence(self,beam):
        """
        Get the start and end node numbers of a beam.

        Parameters
        ----------
        beam : int
            Beam number.

        Returns
        -------
        tuple of int
            (start_node, end_node)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> start, end = staad_obj.Geometry.GetMemberIncidence(1)
        >>> print(f"Start Node: {start}, End Node: {end}")
        """
        safe_n1 = make_safe_array_long(1)
        x = make_variant_vt_ref(safe_n1,  automation.VT_I4)

        safe_n2 = make_safe_array_long(1)
        y = make_variant_vt_ref(safe_n2,  automation.VT_I4)

        result = self._geometry.GetMemberIncidence(beam,x,y)
        if result < 0:
            raise_os_error_if_error_code(result)

        return (x[0],y[0])
```

[docs]

```
    def GetNoOfSelectedBeams(self):
        """
        Get the number of selected beams.

        Returns
        -------
        int
            Number of selected beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> selected_beam_count = staad_obj.Geometry.GetNoOfSelectedBeams()
        >>> print(selected_beam_count)
        """
        return self._geometry.GetNoOfSelectedBeams()
```

[docs]

```
    def GetSelectedBeams(self):
        """
        Get the list of selected beam numbers.

        Returns
        -------
        list of int
            Selected beam numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> selected = staad_obj.Geometry.GetSelectedBeams()
        >>> print(selected)
        """
        n_beams = self._geometry.GetNoOfSelectedBeams()
        safe_list = make_safe_array_long(n_beams)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetSelectedBeams(lista)

        return (lista[0])
```

[docs]

```
    def GetNoOfBeamsConnectedAtNode(self,node):
        """
        Get the number of beams connected at a node.

        Parameters
        ----------
        node : int
            Node number.

        Returns
        -------
        int
            Number of beams connected at the node.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> connected_beam_count = staad_obj.Geometry.GetNoOfBeamsConnectedAtNod
        >>> print(connected_beam_count)
        """
        return self._geometry.GetNoOfBeamsConnectedAtNode(node)
```

[docs]

```
    def GetBeamsConnectedAtNode(self,node):
        """
        Get the list of beams connected at a node.

        Parameters
        ----------
        node : int
            Node number.

        Returns
        -------
        list of int
            Beam numbers connected at the node.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> beams = staad_obj.Geometry.GetBeamsConnectedAtNode(1)
        >>> print(beams)
        """
        No_Nodes = self.GetNoOfBeamsConnectedAtNode(node)

        safe_list = make_safe_array_long(No_Nodes)
        list = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automation

        retval=self._geometry.GetBeamsConnectedAtNode(node,list)

        return list[0]



    ## GROUP FUNCTIONS
```

[docs]

```
    def GetGroupEntityCount(self,group_name):
        """
        Get the number of entities in a group.

        Parameters
        ----------
        group_name : str
            Name of the group.

        Returns
        -------
        int
            Number of entities in the group.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetGroupEntityCount("Group1")
        >>> print(count)
        """
        return self._geometry.GetGroupEntityCount(group_name)
```

[docs]

```
    def GetGroupEntities(self,group_name):
        """
        Get the list of entities in a group.

        Parameters
        ----------
        group_name : str
            Name of the group.

        Returns
        -------
        list of int
            Entity numbers in the group.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> entities = staad_obj.Geometry.GetGroupEntities("Group1")
        >>> print(entities)
        """
        beams = self._geometry.GetGroupEntityCount(group_name)
        safe_list = make_safe_array_long(beams)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        result = self._geometry.GetGroupEntities(group_name,lista)

        if result < 0:
            raise_os_error_if_error_code(result)

        return lista[0]
```

[docs]

```
    def ClearMemberSelection(self):
        """
        Clear the current member selection.
        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.ClearMemberSelection()
        """
        self._geometry.ClearMemberSelection()
```

[docs]

```
    def SelectMultipleBeams(self, beam_ids: list):
        """
        Select multiple beams.

        Parameters
        ----------
        beam_ids : list of int
            List of beam numbers to select.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SelectMultipleBeams([1, 2, 3])
        """
        safe_list = make_safe_array_long_input(beam_ids)
        lista_variant = make_variant_vt_ref(safe_list, automation.VT_ARRAY | au

        self._geometry.SelectMultipleBeams(lista_variant)
```

[docs]

```
    def GetGroupCount(self,grouptype):
        """
        Get the number of groups of a given type.

        Parameters
        ----------
        grouptype : int
            +-------+---------------------------------------+
            | Index | Group Type                            |
            +=======+=======================================+
            | 1     | Nodes                                 |
            +-------+---------------------------------------+
            | 2     | Members                               |
            +-------+---------------------------------------+
            | 3     | Plates                                |
            +-------+---------------------------------------+
            | 4     | Solids                                |
            +-------+---------------------------------------+
            | 5     | Geometry (Members, Plates and Solids) |
            +-------+---------------------------------------+
            | 6     | Floor (Floor beam)                    |
            +-------+---------------------------------------+

        Returns
        -------
        int
            Number of groups.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetGroupCount(1) # For Node groups
        >>> print(count)
        """
        return self._geometry.GetGroupCount(grouptype)
```

[docs]

```
    def GetGroupNames(self,grouptype):
        """
        Get the names of all groups of a given type.

        Parameters
        ----------
        grouptype : int
            +-------+---------------------------------------+
            | Index | Group Type                            |
            +=======+=======================================+
            | 1     | Nodes                                 |
            +-------+---------------------------------------+
            | 2     | Members                               |
            +-------+---------------------------------------+
            | 3     | Plates                                |
            +-------+---------------------------------------+
            | 4     | Solids                                |
            +-------+---------------------------------------+
            | 5     | Geometry (Members, Plates and Solids) |
            +-------+---------------------------------------+
            | 6     | Floor (Floor beam)                    |
            +-------+---------------------------------------+

        Returns
        -------
        list of str
```

```
            List of group names.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> names = staad_obj.Geometry.GetGroupNames(1)
        >>> print(names)
        """
        group_count = self._geometry.GetGroupCount(grouptype)
        group_names_safe_array = make_safe_array_string(group_count)
        group_names = make_variant_vt_ref(group_names_safe_array, automation.VT_

        self._geometry.GetGroupNames(grouptype, group_names)

        return list(group_names[0])



    def CreatePhysicalMember(self,member_list:list):
        """
        Create a physical member from specified analytical members.

        Parameters
        ----------
        member_list : list of int
            List of analytical member IDs to form the physical member.

        Returns
        -------
        int
            ID of the newly created physical member (0 if unsuccessful).

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> pmem_id = staad_obj.Geometry.CreatePhysicalMember([1, 2, 3])
        >>> print(pmem_id)
        """
        num=len(member_list)

        safe_MemberList = make_safe_array_long_input(member_list)
        PhysicalMemID=self._geometry.CreatePhysicalMember(num,safe_MemberList,N
        return PhysicalMemID
```

[docs]

```
    def CreateNode(self, nNodeNo : int, x : float, y : float, z : float):
        """
        Create a node with specified coordinates and node number.

        Parameters
        ----------
        nNodeNo : int
            Node number ID to assign.
        x : float
            X coordinate.
        y : float
            Y coordinate.
        z : float
            Z coordinate.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateNode(10, 0.0, 0.0, 0.0)
        """
        nNodeNo_vt = make_variant_vt(nNodeNo)
        x_vt = make_variant_vt(x)
        y_vt = make_variant_vt(y)
        z_vt = make_variant_vt(z)
        self._geometry.CreateNode(nNodeNo_vt, x_vt, y_vt, z_vt)
```

[docs]

```
    def CreateBeam(self, nBeamNo : int, nNodeStart : int, nNodeEnd : int):
        """
        Create a beam/member with specified nodes.

        Parameters
        ----------
        nBeamNo : int
            Member number ID to assign.
        nNodeStart : int
            ID of the starting node.
        nNodeEnd : int
            ID of the ending node.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateBeam(5, 1, 2)
        """
        nBeamNo_vt = make_variant_vt(nBeamNo)
        nNodeStart_vt = make_variant_vt(nNodeStart)
        nNodeEnd_vt = make_variant_vt(nNodeEnd)
        self._geometry.CreateBeam(nBeamNo_vt, nNodeStart_vt, nNodeEnd_vt)
```

[docs]

```
def CreatePlate(self, nPlateNo : int, nNodeA : int, nNodeB : int, nNodeC :
    """
    Create a plate with specified nodes.

    Parameters
    ----------
    nPlateNo : int
        Plate number ID to assign.
    nNodeA : int
        Node A for plate connectivity.
    nNodeB : int
        Node B for plate connectivity.
    nNodeC : int
        Node C for plate connectivity.
    nNodeD : int
        Node D for plate connectivity.

    Returns
    -------
    None

    Examples
    --------
    >>> from openstaadpy import os_analytical
    >>> staad_obj = os_analytical.connect()
    >>> staad_obj.Geometry.CreatePlate(1, 1, 2, 3, 4)
    """
    nPlateNo_vt = make_variant_vt(nPlateNo)
    nNodeA_vt = make_variant_vt(nNodeA)
    nNodeB_vt = make_variant_vt(nNodeB)
    nNodeC_vt = make_variant_vt(nNodeC)
    nNodeD_vt = make_variant_vt(nNodeD)
    self._geometry.CreatePlate(nPlateNo_vt, nNodeA_vt, nNodeB_vt, nNodeC_vt
```

[docs]

```
def DeleteNode(self, nNodeNo: int):
    """
    Delete a specified node.

    Parameters
    ----------
    nNodeNo : int
        Node number to delete.

    Returns
    -------
    None

    Examples
    --------
```

```
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.DeleteNode(10)
        """
        nNodeNo_vt = make_variant_vt(nNodeNo)
        self._geometry.DeleteNode(nNodeNo_vt)
```

[docs]

```
    def DeleteBeam(self, BeamNo: int):
        """
        Delete a specified beam.

        Parameters
        ----------
        BeamNo : int
            Beam number to delete.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.DeleteBeam(5)
        """
        nBeamNo_vt = make_variant_vt(BeamNo)
        self._geometry.DeleteBeam(nBeamNo_vt)
```

[docs]

```
    def DeletePlate(self, nPlateNo: int):
        """
        Delete a specified plate.

        Parameters
        ----------
        nPlateNo : int
            Plate number to delete.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.DeletePlate(1)
        """
        nPlateNo_vt = make_variant_vt(nPlateNo)
        self._geometry.DeletePlate(nPlateNo_vt)
```

[docs]

```
    def AddNode(self, x : float, y : float, z : float):
        """
        Add a node with specified coordinates and return the assigned node numbe

        Parameters
        ----------
        x : float
            X coordinate.
        y : float
            Y coordinate.
        z : float
            Z coordinate.

        Returns
        -------
        int
            Node number assigned.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> node_no = staad_obj.Geometry.AddNode(0.0, 0.0, 0.0)
        >>> print(node_no)
        """
        result = self._geometry.AddNode(x, y, z)
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def AddBeam(self, nNodeStart : int, nNodeEnd : int):
        """
        Add a beam/member with specified nodes and return the assigned beam numb

        Parameters
        ----------
        nNodeStart : int
            ID of the starting node.
        nNodeEnd : int
            ID of the ending node.

        Returns
        -------
        int
            Beam number assigned.
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> beam_no = staad_obj.Geometry.AddBeam(1, 2)
        >>> print(beam_no)
        """
        nNodeStart_vt = make_variant_vt(nNodeStart)
        nNodeEnd_vt = make_variant_vt(nNodeEnd)
        BeamNo_vt = self._geometry.AddBeam(nNodeStart_vt, nNodeEnd_vt)
        if BeamNo_vt < 0:
            raise_os_error_if_error_code(BeamNo_vt)
        return BeamNo_vt * 1
```

[docs]

```
    def AddPlate(self, nNodeA : int, nNodeB : int, nNodeC : int, nNodeD : int =
        """
        Add a plate with specified nodes and return the assigned plate number.

        Parameters
        ----------
        nNodeA : int
            Node A for plate connectivity.
        nNodeB : int
            Node B for plate connectivity.
        nNodeC : int
            Node C for plate connectivity.
        nNodeD : int
            Node D for plate connectivity.

        Returns
        -------
        int
            Plate number assigned.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> plate_no = staad_obj.Geometry.AddPlate(1, 2, 3, 4)
        >>> print(plate_no)
        """
        nNodeA_vt = make_variant_vt(nNodeA)
        nNodeB_vt = make_variant_vt(nNodeB)
        nNodeC_vt = make_variant_vt(nNodeC)
        nNodeD_vt = make_variant_vt(nNodeD)
        PlateNo_vt = self._geometry.AddPlate(nNodeA_vt, nNodeB_vt, nNodeC_vt, n
        if PlateNo_vt < 0:
            raise_os_error_if_error_code(PlateNo_vt)
        return PlateNo_vt * 1
```

[docs]

```
def SplitBeamInEqlParts(self, nBeamNo: int, nParts: int):
    """
    Split a beam into equal parts.

    Parameters
    ----------
    nBeamNo : int
        Beam number to split.
    nParts : int
        Number of equal parts to split the beam into.

    Returns
    -------
    None

    Examples
    --------
    >>> from openstaadpy import os_analytical
    >>> staad_obj = os_analytical.connect()
    >>> staad_obj.Geometry.SplitBeamInEqlParts(1, 3)
    """
    self._geometry.SplitBeamInEqlParts(nBeamNo, nParts)
```

[docs]

```
def GetLastPlateNo(self):
    """
    Get the last plate number.

    Returns
    -------
    int
        Last plate number.

    Examples
    --------
    >>> from openstaadpy import os_analytical
    >>> staad_obj = os_analytical.connect()
    >>> last_plate = staad_obj.Geometry.GetLastPlateNo()
    >>> print(last_plate)
    """
    result = self._geometry.GetLastPlateNo()
    if result <= 0:
        raise_os_error_if_error_code(-1)
    return result
```

[docs]

```
def SelectPlate(self, nPlateNo: int):
        """
        Select a plate by its number.

        Parameters
        ----------
        nPlateNo : int
            Plate number to select.

        Returns
        -------
        bool
            Status of selection
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectPlate(1)
        >>> print(result) # True if selected, False otherwise
        """
        if nPlateNo not in self.GetPlateList():
            return False
        nPlateNo_vt = make_variant_vt(nPlateNo)
        result = self._geometry.SelectPlate(nPlateNo_vt)
        return bool(result)
```

[docs]

```
    def CreateGroup(self, group_type: int, group_name: str):
        """
        Create a new group with the specified name and type.

        Parameters
        ----------
        group_type : int
            Type of the group:
                +-------+----------------------------------------+
                | Index | Group Type                             |
                +=======+========================================+
                | 1     | Nodes                                  |
                +-------+----------------------------------------+
                | 2     | Members                                |
                +-------+----------------------------------------+
                | 3     | Plates                                 |
                +-------+----------------------------------------+
                | 4     | Solids                                 |
                +-------+----------------------------------------+
                | 5     | Geometry (Members, Plates and Solids) |
                +-------+----------------------------------------+
                | 6     | Floor (Floor beam)                     |
                +-------+----------------------------------------+

        group_name : str
            Name of the group to create.
        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateGroup(1, "MyGroup")
        """
        result = self._geometry.CreateGroup(group_type, group_name)
        if result < 0:
            raise_os_error_if_error_code(result)
```

[docs]

```
    def ClearPlateSelection(self):
        """
        Clear the current plate selection.

        Returns
        -------
        None
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.ClearPlateSelection()
        """
        self._geometry.ClearPlateSelection()
```

[docs]

```
    def CreateSolid(self, solidNo : int, nodeA : int, nodeB : int, nodeC : int,
        """
        Create a solid element.

        Parameters
        ----------
        solidNo : int
            Solid number ID to assign.
        nodeA : int
            ID of node A.
        nodeB : int
            ID of node B.
        nodeC : int
            ID of node C.
        nodeD : int
            ID of node D.
        nodeE : int
            ID of node E.
        nodeF : int
```

```
            ID of node F.
        nodeG : int
            ID of node G.
        nodeH : int
            ID of node H.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateSolid(4, 1, 2, 3, 4, 5, 6, 7, 8)
        """
        self._geometry.CreateSolid(solidNo, nodeA, nodeB, nodeC, nodeD, nodeE,
```

[docs]

```
    def AddSolid(self, nodeA : int, nodeB : int, nodeC : int, nodeD : int, node
        """
        Add a solid element.

        Parameters
        ----------
        nodeA : int
            ID of node A.
        nodeB : int
            ID of node B.
        nodeC : int
            ID of node C.
        nodeD : int
            ID of node D.
        nodeE : int
            ID of node E.
        nodeF : int
            ID of node F.
        nodeG : int
            ID of node G.
        nodeH : int
            ID of node H.

        Returns
        -------
        Int
            ID number of the added solid.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> solidID = staad_obj.Geometry.AddSolid(1, 2, 3, 4, 5, 6, 7, 8)
        >>> print(solidID)
        """
        result = self._geometry.AddSolid(nodeA, nodeB, nodeC, nodeD, nodeE, node
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def AddMultipleNodes(self, coordinates):
        """
        Add multiple nodes at once.

        Parameters
        ----------
        coordinates : list of lists containing float or int
            List of lists containing x, y, z coordinates for each node. [[x1, y1

        Returns
        -------
        List: List of node numbers assigned to the added nodes.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> node_ids = staad_obj.Geometry.AddMultipleNodes([[0.0,0.0,0.0],[1.0,1
        >>> print(node_ids)
        """
        if (
            not isinstance(coordinates, list)
            or not all(isinstance(node, list) for node in coordinates)
            or not all(all(isinstance(coordinate, (float, int)) for coordinate
        ):
            return
        if not all(len(lst) == 3 for lst in coordinates):
            return
        node_ids = []
        for coordinate in coordinates:
            x, y, z = coordinate
            node_id = self.AddNode(x, y, z)
            node_ids.append(node_id)
        return node_ids
```

[docs]

```
    def AddMultipleBeams(self, incidences):
        """
        Add multiple beams at once.

        Parameters
        ----------
        incidences : list of lists containing int
```

```
            List of lists containing start and end node numbers for each beam. [

        Returns
        -------
        List
            List of beam numbers assigned to the added beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> beam_ids = staad_obj.Geometry.AddMultipleBeams([[1,2],[2,3]])
        >>> print(beam_ids)
        """
        if (
            not isinstance(incidences, list)
            or not all(isinstance(beam, list) for beam in incidences)
            or not all(all(isinstance(endnode, int) for endnode in beam) for be
        ):
            return
        if not all(len(lst) == 2 for lst in incidences):
            return
        beam_ids = []
        for incidence in incidences:
            start_node, end_node = incidence
            beam_id = self.AddBeam(start_node, end_node)
            beam_ids.append(beam_id)
        return beam_ids
```

[docs]

```
    def AddMultiplePlates(self, incidences):
        """
        Add multiple plates at once.

        Parameters
        ----------
        incidences : list
            List of lists containing nodeA, nodeB, nodeC, nodeD for each plate.

        Returns
        -------
        List: List of plate numbers assigned to the added plates.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> plate_ids = staad_obj.Geometry.AddMultiplePlates([[1,2,3,4],[2,3,4,
        >>> print(plate_ids)
        """
        if (not isinstance(incidences, list) or
            not all(isinstance(row, list) for row in incidences) or
            not all(all(isinstance(node, int) for node in row) for row in incide
            ):
            return
        if not all(len(lst) in (4,3) for lst in incidences):
            return
        plate_ids = []
        for incidence in incidences:
            if len(incidence) == 3:
                incidence.append(0)
            nodeA, nodeB, nodeC, nodeD = incidence
            plate_id = self.AddPlate(nodeA, nodeB, nodeC, nodeD)
            plate_ids.append(plate_id)
        return plate_ids
```

[docs]

```
    def AddMultipleSolids(self, incidences):
        """
        Add multiple solids at once.

        Parameters
        ----------
        incidences : list
            List of lists containing nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, n

        Returns
        -------
        List: List of solid numbers assigned to the added solids.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> solid_ids = staad_obj.Geometry.AddMultipleSolids([[1,2,3,4,5,6,7,8],
        >>> print(solid_ids)
        """
        if (not isinstance(incidences, list) or
            not all(isinstance(row, list) for row in incidences) or
            not all(all(isinstance(node, int) for node in row) for row in incide
            ):
            return
        if not all(len(lst) in (8,7,6) for lst in incidences):
            return
        solid_ids = []
        for incidence in incidences:
            if len(incidence) == 6:
                incidence.extend([0, 0])
            elif len(incidence) == 7:
                incidence.append(0)
            nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, nodeG, nodeH = incidence
            solid_id = self.AddSolid(nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, n
            solid_ids.append(solid_id)
        return solid_ids
```

[docs]

```
    def DeleteSolid(self, solidID):
        """
        Delete a specified solid.

        Parameters
        ----------
        solidID : int
            ID of solid to delete.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.DeleteSolid(1)
        """
        self._geometry.DeleteSolid(solidID)
```

[docs]

```
    def SplitBeam(self,  beamNo:int, nodes:int, distToNodes:list):
        """
        Split a beam into parts.

        Parameters
        ----------
        beamNo : int
            Beam ID to split.
        nodes : int
            The number of node(s) to be inserted in the beam.
        distToNodes : list
            List of distances in from the start of the beam to each new node.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SplitBeam(1, 2, [1.0, 2.0])
        """
        vt_distToNodes = make_safe_array_double_input(distToNodes)
        self._geometry.SplitBeam(beamNo, nodes, vt_distToNodes)
```

[docs]

```
    def GetLastSolidNo(self):
        """
        Returns the solid number of the last solid created in the model.

        Returns
        -------
        int
            Last solid number.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> last_solid = staad_obj.Geometry.GetLastSolidNo()
        """
        return int(self._geometry.GetLastSolidNo())
```

[docs]

```
    def GetNoOfSelectedPlates(self):
        """
        Return the number of selected plates.

        Returns
        -------
        int
            Number of selected plates.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> selected_plate_count = staad_obj.Geometry.GetNoOfSelectedPlates()
        >>> print(selected_plate_count)
        """
        return int(self._geometry.GetNoOfSelectedPlates())
```

[docs]

```
    def GetSelectedPlates(self, isSorted:bool = False):
        """
        return a list of selected plate numbers.

        Parameters
        ----------
        isSorted : bool optional
            If True, the plate numbers will be sorted. The default is False. (ir

        Returns
        -------
        list of int
            Selected plate numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> plates = staad_obj.Geometry.GetSelectedPlates(True)
        >>> print(plates)
        """
        size = self.GetNoOfSelectedPlates()
        vt_plates = make_safe_array_long(size)
        vt_plates_ref = make_variant_vt_ref(vt_plates, automation.VT_ARRAY | au
        self._geometry.GetSelectedPlates(vt_plates_ref, isSorted)
        return list(vt_plates[0])
```

[docs]

```
    def GetNoOfSelectedSolids(self):
        """
        Get the number of selected solids.

        Returns
        -------
        int
            Number of selected solids.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> n_solids = staad_obj.Geometry.GetNoOfSelectedSolids()
        >>> print(n_solids)
        """
        return int(self._geometry.GetNoOfSelectedSolids())
```

[docs]

```
    def GetSelectedSolids(self, isSorted:bool = False):
        """
        Get the list of selected solid numbers.

        Parameters
        ----------
        isSorted : bool optional
            If True, the solid numbers will be sorted. The default is False. (ir

        Returns
        -------
        list of int
            Selected solid numbers.
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> solids = staad_obj.Geometry.GetSelectedSolids(True)
        >>> print(solids)
        """
        size = self.GetNoOfSelectedSolids()
        vt_solids = make_safe_array_long(size)
        vt_solids_ref = make_variant_vt_ref(vt_solids, automation.VT_ARRAY | au
        self._geometry.GetSelectedSolids(vt_solids_ref, isSorted)
        return list(vt_solids[0])
```

[docs]

```
    def SelectMultipleNodes(self, nodes:list):
        """
        Select multiple nodes.

        Parameters
        ----------
        nodes : list
            node numbers to select.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectMultipleNodes([1,2,3])
        >>> print(result)
        """
        vt_nodes = make_safe_array_long_input(nodes)
        return bool(self._geometry.SelectMultipleNodes(vt_nodes))
```

[docs]

```
    def SelectMultiplePlates(self, plates:list):
        """
        Select multiple plates.

        Parameters
        ----------
        plates : list
            Plate numbers to select.

        Returns
        -------
        bool
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectMultiplePlates([1,2,3])
        >>> print(result)
        """
        vt_plates = make_safe_array_long_input(plates)
        return bool(self._geometry.SelectMultiplePlates(vt_plates))
```

[docs]

```
    def SelectMultipleSolids(self, solids:list):
        """
        Select multiple solids.

        Parameters
        ----------
        solids : list
            Solid numbers to select.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectMultipleSolids([1,2,3])
        >>> print(result)
        """
        vt_solid = make_safe_array_long_input(solids)
        return bool(self._geometry.SelectMultipleSolids(vt_solid))
```

[docs]

```
    def SelectNode(self, nodeID):
        """
        Select a node.

        Parameters
        ----------
        nodeID : int
            node number to select.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectNode(1)
        >>> print(result)
        """
        return bool(self._geometry.SelectNode(nodeID))
```

[docs]

```
    def SelectBeam(self, beamID):
        """
        Select a beam.

        Parameters
        ----------
        beamID : int
            beam number to select.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectBeam(1)
        >>> print(result)
        """
        return bool(self._geometry.SelectBeam(beamID))
```

[docs]

```
    def SelectSolid(self, solidID):
        """
        Select a solid.

        Parameters
        ----------
        solidID : int
            solid number to select.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.SelectSolid(1)
        >>> print(result)
        """
        return bool(self._geometry.SelectSolid(solidID))
```

[docs]

```
    def GetPlateCount(self):
        """
        Returns the number of plates.

        Returns
        -------
        int
            Number of plates.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> plate_count = staad_obj.Geometry.GetPlateCount()
        >>> print(plate_count)
        """
        return int(self._geometry.GetPlateCount())
```

[docs]

```
    def GetSolidCount(self):
        """
        Returns the number of solids.

        Returns
        -------
        int
            Number of solids.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> solid_count = staad_obj.Geometry.GetSolidCount()
        >>> print(solid_count)
        """
        return int(self._geometry.GetSolidCount())
```

[docs]

```
    def GetPlateList(self):
        """
        Returns the list of all plate numbers.
        Returns
        -------
        list of int
            List of plate numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> plates = staad_obj.Geometry.GetPlateList()
        >>> print(plates)
        """
        n_plates =  int(self._geometry.GetPlateCount())
        if n_plates <= 0:
            return []
        safe_list = make_safe_array_long(n_plates)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetPlateList(lista)

        return list(lista[0])
```

[docs]

```
    def GetSolidList(self):
        """
        Get the list of all solid numbers.

        Returns
        -------
        list of int
            List of solid numbers.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> solids = staad_obj.Geometry.GetSolidList()
        >>> print(solids)
        """
        n_solids =  int(self._geometry.GetSolidCount())
        if n_solids <= 0:
            return []
        safe_list = make_safe_array_long(n_solids)
        lista = make_variant_vt_ref(safe_list,  automation.VT_ARRAY | automatio

        self._geometry.GetSolidList(lista)

        return list(lista[0])
```

[docs]

```
    def GetPlateIncidence(self, plateNo:int):
        """
        Get the node incidences A, B, C, D for a plate.

        Parameters
        ----------
        plateNo : int
            Plate number.

        Returns
        -------
        tuple of int
            4 end node IDs for the plate (A, B, C, D).

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> a, b, c, d = staad_obj.Geometry.GetPlateIncidence(1)
        >>> print(a, b, c, d)
        """
        safe_n1 = make_safe_array_long(1)
        vt_A = make_variant_vt_ref(safe_n1,  automation.VT_I4)

        safe_n2 = make_safe_array_long(1)
        vt_B = make_variant_vt_ref(safe_n2,  automation.VT_I4)

        safe_n3 = make_safe_array_long(1)
        vt_C = make_variant_vt_ref(safe_n3,  automation.VT_I4)

        safe_n4 = make_safe_array_long(1)
        vt_D = make_variant_vt_ref(safe_n4,  automation.VT_I4)

        retval = int(self._geometry.GetPlateIncidence(plateNo, vt_A, vt_B, vt_C
        if retval != 0:
            raise Exception(f"Error retrieving plate incidence: {retval}")
        return (vt_A[0], vt_B[0], vt_C[0], vt_D[0])
```

[docs]

```
    def GetSolidIncidence(self, solidNo):
        """
        Get the node incidences for a solid.

        Returns
        -------
        tuple of int
            8 end node IDs for the solid. (A, B, C, D, E, F, G, H)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> nodes = staad_obj.Geometry.GetSolidIncidence(1)
        >>> print(nodes)
        """
        safe_n1 = make_safe_array_long(1)
        vt_A = make_variant_vt_ref(safe_n1,  automation.VT_I4)
        safe_n2 = make_safe_array_long(1)
        vt_B = make_variant_vt_ref(safe_n2,  automation.VT_I4)
        safe_n3 = make_safe_array_long(1)
        vt_C = make_variant_vt_ref(safe_n3,  automation.VT_I4)
        safe_n4 = make_safe_array_long(1)
        vt_D = make_variant_vt_ref(safe_n4,  automation.VT_I4)
        safe_n5 = make_safe_array_long(1)
        vt_E = make_variant_vt_ref(safe_n5,  automation.VT_I4)
        safe_n6 = make_safe_array_long(1)
        vt_F = make_variant_vt_ref(safe_n6,  automation.VT_I4)
        safe_n7 = make_safe_array_long(1)
        vt_G = make_variant_vt_ref(safe_n7,  automation.VT_I4)
        safe_n8 = make_safe_array_long(1)
        vt_H = make_variant_vt_ref(safe_n8,  automation.VT_I4)

        retval = int(self._geometry.GetSolidIncidence(solidNo, vt_A, vt_B, vt_C
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return (vt_A[0], vt_B[0], vt_C[0], vt_D[0], vt_E[0], vt_F[0], vt_G[0], 
```

[docs]

```
    def ClearNodeSelection(self):
        """
        Clear the current node selection.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.ClearNodeSelection()
        """
        self._geometry.ClearNodeSelection()
```

[docs]

```
    def ClearSolidSelection(self):
        """
        Clear the current solid selection.

        Returns
        -------
        None
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.ClearSolidSelection()
        """
        self._geometry.ClearSolidSelection()
```

[docs]

```
    def SetNodeUniqueID(self, nodeNo:int, uniqueID:str):
        """
        Set a unique ID for a node.

        Parameters
        ----------
        nodeNo : int
            Node number to set the unique ID for.
        uniqueID : str
            Unique identifier for the node.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetNodeUniqueID(1, "node-uuid")
        """
        self._geometry.SetNodeUniqueID(nodeNo, uniqueID)
```

[docs]

```
    def SetMemberUniqueID(self, beamNo:int, uniqueID:str):
        """
        Set a unique ID for a member.

        parameters
        ----------
        beamNo : int
            Beam number to set the unique ID for.
        uniqueID : str
            unique identifier for the member.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetMemberUniqueID(1, "beam-uuid")
        """
        self._geometry.SetMemberUniqueID(beamNo, uniqueID)
```

[docs]

```
    def SetPlateUniqueID(self, plateNo:int, uniqueID:str):
        """
        Set a unique ID for a plate.

        Parameters
        ----------
        plateNo : int
            plate number to set the unique ID for.
        uniqueID : str
            unique identifier for the plate.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetPlateUniqueID(1, "plate-uuid")
        """
        self._geometry.SetPlateUniqueID(plateNo, uniqueID)
```

[docs]

```
    def SetSolidUniqueID(self, solidNo:int, uniqueID:str):
        """
        Set a unique ID for a solid.

        Parameters
        ----------
        solidNo : int
            Solid number to set the unique ID for.
        uniqueID : str
            Unique identifier for the solid.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetSolidUniqueID(1, "solid-uuid")
        """
        self._geometry.SetSolidUniqueID(solidNo, uniqueID)
```

[docs]

```
    def SetNodeCoordinate(self, nodeNo: int, x: float, y: float, z: float):
        """
        Set the coordinates of a node.

        Parameters
        ----------
        nodeNo : int
        x : float
        y : float
        z : float

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetNodeCoordinate(1, 0.0, 0.0, 0.0)
        """
        self._geometry.SetNodeCoordinate(nodeNo, x, y, z)
```

[docs]

```
    def DoTranslationalRepeat(self, link_bays:bool, open_base:bool, axis_dir:in
        """
        Perform a translational repeat operation.

        Parameters
        ----------
        link_bays : bool
            specifies whether to generate new members between each step in the

        open_base : bool
            specifies not to generate linking members at the base of the structu

        axis_dir : int
            value to specify direction in global axis along which translational

        spacing_list : list[float]
            List of spacing distances.

        no_of_bays : int
            specifies number of generated bays (maximum no of bays that can be
        renumber_bays : bool
            specifies whether to use a user-specified starting number of the mem

        renumber_list : list[int]
            specify starting member numbers for each newly generated bays (lengt

        geometry_only_flag : bool
            specifies whether only geometry data is to be copied (True = Copy ge

        Returns
        -------
        Result : bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.DoTranslationalRepeat(True, True, 3, [10
        >>> print(result) # True if successful, False otherwise
        """
        if not renumber_bays:
            vt_renumber_list = None
        else:
            vt_renumber_list = make_safe_array_long_input(renumber_list)

        vt_spacing_list = make_safe_array_double_input(spacing_list)

        result = self._geometry.DoTranslationalRepeat(int(link_bays), int(open_b
        return bool(result)
```

[docs]

```
    def GetNodeUniqueID(self, nodeNo: int):
        """
        Get the unique ID of a node.

        Parameters
        ----------
        nodeNo : int

        Returns
        -------
        str

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetNodeUniqueID(1)
        >>> print(uid)
        """
        return self._geometry.GetNodeUniqueID(nodeNo)
```

[docs]

```
    def GetMemberUniqueID(self, memberNo: int):
        """
        Get the unique ID of a member.

        Parameters
        ----------
        memberNo : int

        Returns
        -------
        str
            Unique ID of the member.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetMemberUniqueID(1)
        >>> print(uid)
        """
        return self._geometry.GetMemberUniqueID(memberNo)
```

[docs]

```
    def GetPlateUniqueID(self, plateNo: int):
        """
        Get the unique ID of a plate.

        Parameters
        ----------
        plateNo : int

        Returns
        -------
        str

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetPlateUniqueID(1)
        >>> print(uid)
        """
        return self._geometry.GetPlateUniqueID(plateNo)
```

[docs]

```
    def GetSolidUniqueID(self, solidNo: int):
        """
        Get the unique ID of a solid.

        Parameters
        ----------
        solidNo : int

        Returns
        -------
        str

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetSolidUniqueID(1)
        >>> print(uid)
        """
        return self._geometry.GetSolidUniqueID(solidNo)
```

[docs]

```
    def GetPlateNodeCount(self, plateNo: int):
        """
        Get the number of nodes in a plate.

        Parameters
        ----------
        plateNo : int
            Plate number.

        Returns
        -------
        int
            Number of nodes in the plate.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> node_count = staad_obj.Geometry.GetPlateNodeCount(1)
        >>> print(node_count)
        """
        return self._geometry.GetPlateNodeCount(plateNo)
```

[docs]

```
    def GetNoOfGeneratedQuadPanels(self):
        """
        Get the number of generated quad panels for selected beams.

        Returns
        -------
        int
            Number of generated quad panels.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> quad_panel_count = staad_obj.Geometry.GetNoOfGeneratedQuadPanels()
        >>> print(quad_panel_count)
        """
        return self._geometry.GetNoOfGeneratedQuadPanels()
```

[docs]

```
    def GetGeneratedQuadPanelIncidences(self):
        """
        Get the incidences of generated quad panels for selected beams.

        Returns
        -------
        List of lists of int
            List of 4 lists containing NodeAs, NodeBs, NodeCs, NodeDs in respect

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> incidences = staad_obj.Geometry.GetGeneratedQuadPanelIncidences(1)
        """
        size = self.GetNoOfGeneratedQuadPanels()
        if size <= 0:
            return [[], [], [], []]

        list_a = make_safe_array_long(size)
        vt_a = make_variant_vt_ref(list_a,  automation.VT_ARRAY | automation.VT_

        list_b = make_safe_array_long(size)
        vt_b = make_variant_vt_ref(list_b,  automation.VT_ARRAY | automation.VT_

        list_c = make_safe_array_long(size)
        vt_c = make_variant_vt_ref(list_c,  automation.VT_ARRAY | automation.VT_

        list_d = make_safe_array_long(size)
        vt_d = make_variant_vt_ref(list_d,  automation.VT_ARRAY | automation.VT_

        self._geometry.GetGeneratedQuadPanelIncidences(vt_a, vt_b, vt_c, vt_d)
        return [list(vt_a[0]), list(vt_b[0]), list(vt_c[0]), list(vt_d[0])]
```

[docs]

```
    def IsZUp(self):
        """
        Check if the Z axis is up.

        Returns
        -------
        bool
            True if Z is up, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> is_z_up = staad_obj.Geometry.IsZUp()
        >>> print(is_z_up)
        """
        return bool(self._geometry.IsZUp())
```

[docs]

```
    def IsBeam(self, beam_no: int, tol_angle: float):
        """
        Returns True if the angle of inclination for specified BEAM member is no

        Parameters
        ----------
        beam_no : int
        tol_angle : float

        Returns
        -------
        bool
            True if the beam is within the tolerance angle, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> is_beam = staad_obj.Geometry.IsBeam(1, 1)
        >>> print(is_beam)
        """
        result = int(self._geometry.IsBeam(beam_no, tol_angle))
        if result < 0:
            raise_os_error_if_error_code(result)
        return bool(result)
```

[docs]

```
    def IsColumn(self, column_no: int, tol_angle: float):
        """
        Returns True if the angle of inclination for specified COLUMN member is

        Parameters
        ----------
        column_no : int
        tol_angle : float

        Returns
        -------
        bool
            True if the column is within the tolerance angle, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> is_column = staad_obj.Geometry.IsColumn(1, 1)
        >>> print(is_column)
        """
        result = int(self._geometry.IsColumn(column_no, tol_angle))
        if result < 0:
            raise_os_error_if_error_code(result)
        return bool(result)
```

[docs]

```
    def RenumberBeam(self, oldBeamNo: int, newBeamNo: int):
        """
        Renumber a beam.

        Parameters
        ----------
        oldBeamNo : int
        newBeamNo : int

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.RenumberBeam(1, 10)
        >>> print(result)
        """
        result = int(self._geometry.RenumberBeam(oldBeamNo, newBeamNo))
        return bool(result)
```

[docs]

```
    def IsOrphanNode(self, nodeNo: int):
        """
        Check if a node is an orphan.
        Parameters
        ----------
        nodeNo : int

        Returns
        -------
        bool
            True if orphan, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> is_orphan = staad_obj.Geometry.IsOrphanNode(1)
        >>> print(is_orphan)
        """
        return bool(self._geometry.IsOrphanNode(nodeNo))
```

[docs]

```
    def GetGroupCountAll(self):
        """
        Get the total number of groups.

        Returns
        -------
        int
            Total group count.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> total_groups = staad_obj.Geometry.GetGroupCountAll()
        >>> print(total_groups)
        """
        return self._geometry.GetGroupCountAll()
```

[docs]

```
    def CreateGroupEx(self, groupType: int, groupName: str, entityList: list):
        """
        Create a group with extended options.

        Parameters
        ----------
        groupType : int
            The int representing the corresponding group type as show in below
                +-------+----------------------------------------+
                | Index | Group Type                             |
                +=======+========================================+
                | 1     | Nodes                                  |
                +-------+--------------------------------------+
                | 2     | Members                              |
                +-------+--------------------------------------+
                | 3     | Plates                               |
                +-------+--------------------------------------+
                | 4     | Solids                               |
                +-------+--------------------------------------+
                | 5     | Geometry (Members, Plates and Solids) |
                +-------+--------------------------------------+
                | 6     | Floor (Floor beam)                   |
                +-------+--------------------------------------+

        groupName : str
            Name of the group.

        entityList : list of int
            List of entity IDs to include in the group.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateGroupEx(1, "GroupA", [1,2,3])
        """
        size = len(entityList)
        if size == 0:
            raise_os_error_if_error_code(-110)
        vt_entityList = make_safe_array_long_input(entityList)
        result = self._geometry.CreateGroupEx(groupType, groupName, size, vt_en
        if result < 0:
            raise_os_error_if_error_code(result)
```

[docs]

```
    def DeleteGroup(self, groupName: str):
        """
        Delete a group.

        Parameters
        ----------
        groupName : str

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.DeleteGroup("GroupA")
        """
        self._geometry.DeleteGroup(groupName)
```

[docs]

```
    def UpdateGroup(self, groupName: str, update_option:int,entityList: list[in
        """
        Updates (replaces, removes, adds) entities to a specified group.

        Parameters
        ----------
        groupName : str

        update_option : int
            +-------+--------------------+
            | Index | Update Option      |
            +=======+====================+
            | 0     | Replace entities   |
            +-------+--------------------+
            | 1     | Remove entities    |
            +-------+--------------------+
            | 2     | Add entities       |
            +-------+--------------------+

        entityList : list of int

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.UpdateGroup("GroupA", [1,2,3])
        """
        vt_entityList = make_safe_array_long_input(entityList)
        self._geometry.UpdateGroup(groupName, update_option, len(entityList), v
```

[docs]

```
    def DefineParametricSurface(self, name : str, type: int, origin_Node: int,
        """
        Define a parametric surface.

        Parameters
        ----------
        name : str
            Name of the parametric surface.

        type : int
```

```
            Type of the parametric surface:
                +-------+------------------------------+
                | value | Surface Type                 |
                +=======+==============================+
                | 0     | None                         |
                +-------+------------------------------+
                | 1     | Wall                         |
                +-------+------------------------------+
                | 2     | Slab                         |
                +-------+------------------------------+

        origin_Node : int
            Node number defining the origin of the parametric surface.

        x_vertex_node : int
            Node number defining the local X axis of the parametric surface.

        y_vertex_node : int
            Node number defining the local Y axis of the parametric surface.

        vertices_list : list[int]
            List of vertices of the parametric surface. (must lie in same plane)

        auto_generate : bool
            Specifies whether to auto-generate boundary points and density obje

        Returns
        -------
        int
            The ID of the created parametric surface.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> surface_id = staad_obj.Geometry.DefineParametricSurface("Surface1",
        >>> print(surface_id)
        """
        if len(vertices_list) < 0:
            raise OsInvalidArgument()
        vt_vertices_list = make_safe_array_long_input(vertices_list)
        result = self._geometry.DefineParametricSurface(name, type, origin_Node
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def AddParametricSurfaceToModel(self, surfaceNo: int):
        """
        Add definition of the specified parametric surface to the model.

        Parameters
        ----------
        surfaceNo : int

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> surface_id = staad_obj.Geometry.AddParametricSurfaceToModel(1)
        >>> print(surface_id)
        """
        result = self._geometry.AddParametricSurfaceToModel(surfaceNo)
        return bool(result)
```

[docs]

```
    def CommitParametricSurfaceMesh(self, surfaceNo: int):
        """
        Merges the specified parametric mesh with the model

        Parameters
        ----------
        surfaceNo : int
            surface ID of the parametric surface to be merged with the model.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.CommitParametricSurfaceMesh(1)
        >>> print(result)
        """
        result = self._geometry.CommitParametricSurfaceMesh(surfaceNo)
        return bool(result)
```

[docs]

```
    def RemoveParametricSurfaceMesh(self, surfaceNo: int):
        """
        Remove the specified parametric mesh from the model.

        Parameters
        ----------
        surfaceNo : int
            surface ID of the parametric surface to be delete from model.
        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.RemoveParametricSurfaceMesh(1)
        >>> print(result)
        """
        result = self._geometry.RemoveParametricSurfaceMesh(surfaceNo)
        return bool(result)
```

[docs]

```
    def AddDensityPointToSurface(self, surfaceNo: int, pointData):
        """
        Add a density point to a surface.

        Parameters
        ----------
        surfaceNo : int
        pointData : object

        Returns
        -------
        None
        """
        self._geometry.AddDensityPointToSurface(surfaceNo, pointData)
```

[docs]

```
    def AddDensityLineToSurface(self, surfaceNo: int, x1: float, y1: float, z1:
        """
        Add a density line to a surface.

        Parameters
        ----------
        surfaceNo : int
            Surface ID of the parametric surface to which the density line will
        x1 : float
            Global X coordinate of the start point of the density line.
        y1 : float
            Global Y coordinate of the start point of the density line.
        z1 : float
            Global Z coordinate of the start point of the density line.
        density1 : int
            Density at the start point of the density line.
        x2 : float
            Global X coordinate of the end point of the density line.
        y2 : float
            Global Y coordinate of the end point of the density line.
        z2 : float
            Global Z coordinate of the end point of the density line.
        density2 : int
            Density at the end point of the density line.
        divisions : int
            Number of divisions along the density line.

        Returns
        -------
        int
            index (0 based) of the density line added.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.AddDensityLineToSurface(1, 0.0, 0.0, 0.0
        >>> print(result)
        """
        result = self._geometry.AddDensityLineToSurface(surfaceNo, x1, y1, z1, d
        if result < 0:
            raise_os_error_if_error_code(result)
        return result
```

[docs]

```
    def AddCircularRegionToSurface(self, surfaceNo: int, x: float, y: float, z:
        """
        Add a circular region or opening to a surface.

        Parameters
        ----------
        surfaceNo : int
            Surface ID of the parametric surface to which the circular region wi

        x : float
            Global X coordinate of the center of the circular region.

        y : float
            Global Y coordinate of the center of the circular region.

        z : float
            Global Z coordinate of the center of the circular region.

        radius : float
            Radius of the circular region.

        divisions : int
            Number of divisions along the circular region.

        density : int
            Density of the circular region.
        is_opening : bool
            Whether the circular region is an opening or not.

        Returns
        -------
        bool

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.AddCircularRegionToSurface(1, 5.0, 5.0,
        >>> print(result)
        """
        retval = self._geometry.AddCircularRegionToSurface(surfaceNo, x, y, z,
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return bool(retval)
```

[docs]

```
    def AddPolygonalRegionToSurface(self, surfaceNo: int, regionData):
        """
        Add a polygonal region to a surface.

        Parameters
        ----------
        surfaceNo : int
        regionData : object

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.AddPolygonalRegionToSurface(1, regionData)
        """
        self._geometry.AddPolygonalRegionToSurface(surfaceNo, regionData)
```

[docs]

```
    def GetParametricSurfaceCount(self):
        """
        Get the number of parametric surfaces.

        Returns
        -------
        int
```

```
            Number of parametric surfaces.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> num_surfaces = staad_obj.Geometry.GetParametricSurfaceCount()
        """
        return self._geometry.GetParametricSurfaceCount()
```

[docs]

```
    def GetParametricSurfaceInfoEx(self, surfaceNo: int):
        """
        Get information about a parametric surface.

        Parameters
        ----------
        surfaceNo : int

        Returns
        -------
        tuple containing various details about the surface
            1. Surface Name (str) : Name of the Mesh
            2. Surface Type (int) : (0: None, 1: Wall, 2: Slab)
            3. Surface sub-type (str) : Sub type of the surface
            4. number of vertices (int) : Number of vertices after meshing
            5. Mesh Size (float) : Target mesh size
            6. Divisions (int) : Number of divisions along the boundary
            7. Meshing method (int) : (0: Basic, 1: Advanced)
            8. isQuad (bool) : Whether the mesh is Quad or Triangular (True = Qu
            9. Origin Node (int) : Origin Node ID
            10. X Node (int) : Node ID on X axis to determine x axis
            11. Y Node (int) : Node ID towards positive Y axis
            12. Number of Circular Openings (int) : Number of circular openings
            13. Number of Polygonal Openings (int) : Number of polygonal opening
            14. Number of Circular Regions (int) : Number of circular regions
            15. Number of Polygonal Regions (int) : Number of polygonal regions
            16. Number of Density Points (int) : Number of density points
            17. Number of Density Lines (int) : Number of density lines

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> surface_name, surface_type, surface_sub_type, number_of_vertices, me
        """
        surface_name = make_safe_str()
        surface_name_ref = make_variant_vt_ref(surface_name, automation.VT_BSTR

        sub_type = make_safe_str()
        sub_type_ref = make_variant_vt_ref(sub_type, automation.VT_BSTR)

        type_vt = make_safe_array_long(1)
        type_ref = make_variant_vt_ref(type_vt, automation.VT_I4)

        num_vertices_vt = make_safe_array_long(1)
        num_vertices_ref = make_variant_vt_ref(num_vertices_vt, automation.VT_I

        mesh_size_vt = make_safe_array_double(1)
        mesh_size_ref = make_variant_vt_ref(mesh_size_vt, automation.VT_R8)

        num_divisions_vt = make_safe_array_long(1)
        num_divisions_ref = make_variant_vt_ref(num_divisions_vt, automation.VT_

        meshing_method_vt = make_safe_array_long(1)
        meshing_method_ref = make_variant_vt_ref(meshing_method_vt, automation.

        is_quad_vt = make_safe_array_long(1)
        is_quad_ref = make_variant_vt_ref(is_quad_vt, automation.VT_I4)

        origin_node_vt = make_safe_array_long(1)
        origin_node_ref = make_variant_vt_ref(origin_node_vt, automation.VT_I4)

        x_node_vt = make_safe_array_long(1)
        x_node_ref = make_variant_vt_ref(x_node_vt, automation.VT_I4)

        y_node_vt = make_safe_array_long(1)
        y_node_ref = make_variant_vt_ref(y_node_vt, automation.VT_I4)

        num_circular_openings_vt = make_safe_array_long(1)
        num_circular_openings_ref = make_variant_vt_ref(num_circular_openings_v

        num_polygonal_openings_vt = make_safe_array_long(1)
        num_polygonal_openings_ref = make_variant_vt_ref(num_polygonal_openings_

        num_circular_regions_vt = make_safe_array_long(1)
        num_circular_regions_ref = make_variant_vt_ref(num_circular_regions_vt,

        num_polygonal_regions_vt = make_safe_array_long(1)
        num_polygonal_regions_ref = make_variant_vt_ref(num_polygonal_regions_v

        num_density_points_vt = make_safe_array_long(1)
        num_density_points_ref = make_variant_vt_ref(num_density_points_vt, aut

        num_density_lines_vt = make_safe_array_long(1)
        num_density_lines_ref = make_variant_vt_ref(num_density_lines_vt, autom

        retval = self._geometry.GetParametricSurfaceInfoEx(surfaceNo, surface_na
                                                       num_vertices_ref, mesh_s
                                                       meshing_method_ref, is_q
                                                       x_node_ref, y_node_ref,
                                                       num_polygonal_openings_re
                                                       num_polygonal_regions_re

        if retval <= 0:
            raise_os_error_if_error_code(-1)
        return (surface_name_ref[0], type_ref[0], sub_type_ref[0], num_vertices_
```

[docs]

```
    def GetParametricSurfaceMeshInfo(self, surfaceNo: int):
        """
        Gets information about specified parametric surface available in the cur

        Parameters
        ----------
        surfaceNo : int
            Surface ID of the parametric surface.

        Returns
        -------
        tuple containing mesh details
            1. Node count (int) : Number of nodes in the mesh
            2. Element count (int) : Number of elements in the mesh

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> nodes, elements = staad_obj.Geometry.GetParametricSurfaceMeshInfo(1)
        >>> print(f"Nodes: {nodes}, Elements: {elements}")
        """
        node_count_vt = make_safe_array_long(1)
        node_count_ref = make_variant_vt_ref(node_count_vt, automation.VT_I4)

        element_count_vt = make_safe_array_long(1)
        element_count_ref = make_variant_vt_ref(element_count_vt, automation.VT_

        retval = self._geometry.GetParametricSurfaceMeshInfo(surfaceNo, node_co
        if retval <= 0:
            raise_os_error_if_error_code(-1)
        return (node_count_ref[0], element_count_ref[0])
```

[docs]

```
    def GetParametricSurfaceMeshData(self, surfaceNo: int):
        """
        Gets data about specified parametric surface available in the currently

        Parameters
        ----------
        surfaceNo : int

        Returns
        -------
        tuple containing mesh data
            1. Nodes (list) : List of generated node ids
            2. Elements (list) : List of generated element(plate) ids

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> mesh_data = staad_obj.Geometry.GetParametricSurfaceMeshData(1)
        >>> print(mesh_data)
        """
        node_count, element_count = self.GetParametricSurfaceMeshInfo(surfaceNo
        nodes = make_safe_array_long(node_count)
        nodes_ref = make_variant_vt_ref(nodes,  automation.VT_ARRAY | automatio

        elements = make_safe_array_long(element_count)
        elements_ref = make_variant_vt_ref(elements,  automation.VT_ARRAY | auto

        retval = self._geometry.GetParametricSurfaceMeshData(surfaceNo, nodes_re
        if retval <= 0:
            raise_os_error_if_error_code(-1)

        return (list(nodes_ref[0]), list(elements_ref[0]))
```

[docs]

```
    def SetParametricSurfaceUniqueID(self, surface_name: str, unique_id: str):
        """
        Set a unique ID for a parametric surface.

        Parameters
        ----------
        surface_name : str
        unique_id : str

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetParametricSurfaceUniqueID("SECOND_FLOOR_SLAB",
        """
        retval = self._geometry.SetParametricSurfaceUniqueID(surface_name, uniq
        if retval < 0:
            raise_os_error_if_error_code(retval)
```

[docs]

```
    def GetParametricSurfaceUniqueID(self, surface_name: str):
        """
        Get the unique ID of a parametric surface.

        Parameters
        ----------
        surface_name : str
        Returns
        -------
        str
            Unique ID of the parametric surface.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetParametricSurfaceUniqueID("SECOND_FLOOR_
        >>> print(uid)
        """
        retval = self._geometry.GetParametricSurfaceUniqueID(surface_name)
        return retval
```

[docs]

```
    def GetAreaOfPlates(self, plateList):
        """
        Get the area of plates.

        Parameters
        ----------
        plateList : list of int

        Returns
        -------
        List
            list of area of each plate in the list.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> area_list = staad_obj.Geometry.GetAreaOfPlates([1,2,3])
        >>> print(area_list)
        """
        vt_plateList = make_safe_array_long_input(plateList)
        size = len(plateList)
        if size == 0:
            return []
        vt_area_list = make_safe_array_double(size)
        vt_area_ref = make_variant_vt_ref(vt_area_list,  automation.VT_ARRAY |
        retval = self._geometry.GetAreaOfPlates(vt_plateList, vt_area_ref)
        if retval <= 0:
            raise_os_error_if_error_code(retval)
        return list(vt_area_ref[0])
```

[docs]

```
    def CreateMultiplePlates(self, plate_ids:list | int, plate_incidences: list
        """
        Create multiple plates.

        Parameters
        ----------
        plate_ids : list of int or int
            plate IDs for each plate. [PlateID1, PlateID2, PlateID3, ...]

        plate_incidences : list of lists
            List of lists containing incidences for each plate. [[NodeA1, NodeB1

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateMultiplePlates(plateIds, plateIncidences)
        """
        if isinstance(plate_ids, int):
            plate_ids = [plate_ids]
        if not len(plate_ids) == len(plate_incidences):
            raise_os_error_if_error_code(-100)
        if (not isinstance(plate_incidences, list) or
            not all(isinstance(row, list) for row in plate_incidences) or
            not all(all(isinstance(node, int) for node in row) for row in plate_
            ):
            return
        if not all(len(lst) in (4,3) for lst in plate_incidences):
            return
        for i in range(len(plate_incidences)):
            incidence = plate_incidences[i]
            if len(incidence) == 3:
                incidence.append(0)
            nodeA, nodeB, nodeC, nodeD = incidence
            self.CreatePlate(plate_ids[i], nodeA, nodeB, nodeC, nodeD)
```

[docs]

```
    def SetParametricSurfaceSubType(self, surfaceName: str, subType: str):
        """
        Set the subtype for a parametric surface.

        Parameters
        ----------
        surfaceName : str
            Name of the parametric surface.
        subType : str
            Sub-type of surface.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetParametricSurfaceSubType("SECOND_FLOOR_SLAB",
        """
        self._geometry.SetParametricSurfaceSubType(surfaceName, subType)
```

[docs]

```
    def GetParametricSurfaceSubType(self, surfaceName: str):
        """
        Get the subtype of a parametric surface.

        Parameters
        ----------
        surfaceNo : int

        Returns
        -------
        str : subtype information

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> subtype = staad_obj.Geometry.GetParametricSurfaceSubType("SECOND_FLO
        >>> print(subtype)
        """
        return self._geometry.GetParametricSurfaceSubType(surfaceName)
```

[docs]

```
    def CreateMultipleNodes(self, node_ids: list, nodeCoordinates: list):
        """
        Create multiple nodes.

        Parameters
        ----------
        node_ids : list of int
            Node IDs for each node. [NodeID1, NodeID2, NodeID3, ...]
        nodeCoordinates : list of lists
            List of [x, y, z] coordinates for each node. [[x1, y1, z1], [x2, y2,

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateMultipleNodes(node_ids,nodeCoordinates)
        """
        if not len(node_ids) == len(nodeCoordinates):
            raise_os_error_if_error_code(-100)
        if (not isinstance(nodeCoordinates, list) or
            not all(isinstance(row, list) for row in nodeCoordinates) or
            not all(all(isinstance(coord, (int, float)) for coord in row) for r
            ):
            return
        for i in range(len(nodeCoordinates)):
            coords = nodeCoordinates[i]
            if len(coords) != 3:
                raise_os_error_if_error_code(-100)
            x, y, z = coords
            self.CreateNode(node_ids[i], x, y, z)
```

[docs]

```
    def CreateMultipleBeams(self, beam_ids: list, beam_incidences: list):
        """
        Create multiple beams.

        Parameters
        ----------
        beam_ids : list of int
            Beam IDs for each beam. [BeamID1, BeamID2, BeamID3, ...]
        beam_incidences : list of lists
            List of [start_node, end_node] for each beam. [[start1, end1], [star

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.CreateMultipleBeams(beam_ids, beam_incidences)
        """
        if not len(beam_ids) == len(beam_incidences):
            raise_os_error_if_error_code(-100)
        if (not isinstance(beam_incidences, list) or
            not all(isinstance(row, list) for row in beam_incidences) or
            not all(all(isinstance(node, int) for node in row) for row in beam_
            ):
            return
        for i in range(len(beam_incidences)):
            incidence = beam_incidences[i]
            if len(incidence) != 2:
                raise_os_error_if_error_code(-100)
            start_node, end_node = incidence
            self.CreateBeam(beam_ids[i], start_node, end_node)
```

[docs]

```
def GetParametricSurfaceInfo(self, surfaceNo: int):
    """
    Get extended information about a parametric surface.

    Parameters
    ----------
    surfaceNo : int

    Returns
    -------
    tuple of various details about the surface
        1. Surface Name (str) : Name of the Mesh
        2. Surface Type (str) : Type of the Mesh (e.g., Slab, Wall)
        3. boundary points count (int) : Number of boundary points
        4. density points count (int) : Number of density points
        5. opening count (int) : Number of openings
        6. region count (int) : Number of regions

    Examples
    --------
    >>> from openstaadpy import os_analytical
    >>> staad_obj = os_analytical.connect()
    >>> surface_name, surface_type, boundary_points, density_points, opening
    """
    surface_name = make_safe_str()
    surface_name_ref = make_variant_vt_ref(surface_name, automation.VT_BSTR)

    type_str = make_safe_str()
    type_ref = make_variant_vt_ref(type_str, automation.VT_BSTR)

    boundary_points_count_vt = make_safe_array_long(1)
    boundary_points_count_ref = make_variant_vt_ref(boundary_points_count_v

    density_points_count_vt = make_safe_array_long(1)
    density_points_count_ref = make_variant_vt_ref(density_points_count_vt,

    opening_count_vt = make_safe_array_long(1)
    opening_count_ref = make_variant_vt_ref(opening_count_vt, automation.VT_

    region_count_vt = make_safe_array_long(1)
    region_count_ref = make_variant_vt_ref(region_count_vt, automation.VT_I4

    retval = self._geometry.GetParametricSurfaceInfo(surfaceNo, surface_name
                                               boundary_points_count_re
                                               opening_count_ref, regio
    if retval == 0:
        raise_os_error_if_error_code(-1)
    return (surface_name_ref[0], type_ref[0], boundary_points_count_ref[0],
```

[docs]

```
def IntersectBeams(self, method: int, beamList: list, tolerance: float ):
    """
    Intersect beams.

    Parameters
    ----------
    method : int
        +-------+--------------------+
        | Index | Method             |
        +=======+====================+
        | 1     | Highlight          |
        +-------+--------------------+
        | 2     | Intersect          |
        +-------+--------------------+

    beamList : list of int
        list of beam IDs to intersect. if it is empty, all beams in the mode

    tolerance : float
        Tolerance to be used for finding beam intersection, should not be ne

    Returns
    -------
    List of int
        IDs of the beams that have been changed and added, only used for int

    Examples
    --------
    >>> from openstaadpy import os_analytical
    >>> staad_obj = os_analytical.connect()
    >>> new_Ids = staad_obj.Geometry.IntersectBeams([1,2,3])
    >>> print(new_Ids)
    """
    size = self.GetIntersectBeamsCount(beamList, tolerance)

    if size == 0 or method not in (1,2):
        return []

    vt_newIds = make_safe_array_long(size)
    vt_newIds_ref = make_variant_vt_ref(vt_newIds, automation.VT_ARRAY | au
    vt_beamList = make_safe_array_long_input(beamList)
    retval = self._geometry.IntersectBeams(method, vt_beamList, tolerance, v
    if retval <= 0:
        raise_os_error_if_error_code(-1)
    return list(vt_newIds_ref[0])
```

[docs]

```
def MergeBeams(self, beamList: list, newId: int, property_id: int, beta_ang
    """
    Merge beams.
```

```
        Parameters
        ----------
        beamList : list of int
            List of beam IDs to merge.
        newId : int
            New ID for the merged beam.
        property_id : int
            Property ID to assign to the merged beam.
        beta_angle : float
            Beta angle for the merged beam.
        material_name : str
            Material name for the merged beam.

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.MergeBeams([1,2,3], 4, 5, 30.0, "Steel")
        >>> print(result)
        """
        vt_beamList = make_safe_array_long_input(beamList)
        retval = self._geometry.MergeBeams(vt_beamList, newId, property_id, bet
        return bool(retval)
```

[docs]

```
    def MergeNodes(self, new_Id: int, nodeList: list):
        """
        Merge nodes.

        Parameters
        ----------
        new_Id : int
            New ID for the merged node.
        nodeList : list of int
            List of node IDs to merge.

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.MergeNodes(4, [1,2,3])
        >>> print(result)
        """
        vt_nodeList = make_safe_array_long_input(nodeList)
        retval = self._geometry.MergeNodes(new_Id, vt_nodeList)
        return bool(retval)
```

[docs]

```
    def GetCountOfBreakableBeamsAtSpecificNodes(self, nodeList: list):
        """
        Get number of beams that can be broken based on the list of node Ids.

        Parameters
        ----------
        nodeList : list of int
            List of node IDs to check for breakable beams.

        Returns
        -------
        int
            Count of breakable beams.

        see Also
        --------
        BreakBeamsAtSpecificNodes

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetCountOfBreakableBeamsAtSpecificNodes(
        >>> print(count)
        """
        vt_nodeList = make_safe_array_long_input(nodeList)
        return self._geometry.GetCountOfBreakableBeamsAtSpecificNodes(vt_nodeLi
```

[docs]

```
    def BreakBeamsAtSpecificNodes(self, nodeList: list):
        """
        Breaks beams that passes through the specified list of nodes and assigns

        Parameters
        ----------
        nodeList : list of int
            List of node IDs where beams should be broken.

        Returns
        -------
        tuple of 2 lists
            1. List of int : IDs of the broken beams.
            2. List of int : IDs of the newly created beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> new_beams = staad_obj.Geometry.BreakBeamsAtSpecificNodes([1,2,3])
        >>> print(new_beams)
        """
        vt_nodeList = make_safe_array_long_input(nodeList)
        size = self.GetCountOfBreakableBeamsAtSpecificNodes(nodeList)
        if size == 0:
            return ([], [])
        vt_brokenIds = make_safe_array_long(size)
        vt_brokenIds_ref = make_variant_vt_ref(vt_brokenIds, automation.VT_ARRAY
        vt_newIds = make_safe_array_long(size)
        vt_newIds_ref = make_variant_vt_ref(vt_newIds, automation.VT_ARRAY | au
        retval = self._geometry.BreakBeamsAtSpecificNodes(vt_nodeList, vt_broke
        if retval <= 0:
            raise_os_error_if_error_code(-1)
        return (list(vt_brokenIds_ref[0]), list(vt_newIds_ref[0]))
```

[docs]

```
    def GetIntersectBeamsCount(self, beamList: list, tolerance: float):
        """
        Get the count of intersecting beams.

        Parameters
        ----------
        beamList : list of int
            list of beam IDs to check for intersection. if it is empty, all bea
        tolerance : float
            Tolerance to be used for finding beam intersection, should not be ne

        Returns
        -------
        int
            Number of intersecting beams.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetIntersectBeamsCount([1,2,3])
        """
        vt_beamList = make_safe_array_long_input(beamList)
        return self._geometry.GetIntersectBeamsCount(vt_beamList, tolerance)
```

[docs]

```
    def ClearPhysicalMemberSelection(self):
        """
        Clears the current selection of physical members.
        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.ClearPhysicalMemberSelection()
        """
        self._geometry.ClearPhysicalMemberSelection()
```

[docs]

```
    def CreatePhysicalMember(self, memberList: list):
        """
        Create a physical member from the currently selected members.

        Parameters
        ----------
        memberList : list of int
            List of member IDs to include in the physical member.
        physicalMemberName : str
            Name of the physical member to create.

        Returns
        -------
        int
            Id of the created physical member.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.CreatePhysicalMember([1,2,3], "MyPhysica
        >>> print(result)
        """
        size = len(memberList)
        if size == 0:
            raise_os_error_if_error_code(-100)
        vt_memberList = make_safe_array_long_input(memberList)
        retval = self._geometry.CreatePhysicalMember(size, vt_memberList)
        if retval == 0:
            raise_os_error_if_error_code(-1)
        return int(retval)
```

[docs]

```
    def DeletePhysicalMember(self, physicalMemberId: int):
        """
        Delete a physical member.
        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member to delete.

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> result = staad_obj.Geometry.DeletePhysicalMember(1)
        >>> print(result)
        """
        retval = self._geometry.DeletePhysicalMember(physicalMemberId)
        return bool(retval)
```

[docs]

```
    def GetAnalyticalMemberCountForPhysicalMember(self, physicalMemberId: int):
        """
        Get the count of analytical members in a physical member.

        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member.

        Returns
        -------
        int
            Count of analytical members in the physical member.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetAnalyticalMemberCountForPhysicalMember
        >>> print(count)
        """
        retval = self._geometry.GetAnalyticalMemberCountForPhysicalMember(physic
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return int(retval)
```

[docs]

```
    def GetAnalyticalMembersForPhysicalMember(self, physicalMemberId: int):
        """
        Get the analytical members in a physical member.

        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member.

        Returns
        -------
        list of int
            List of analytical member IDs in the physical member.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> members = staad_obj.Geometry.GetAnalyticalMembersForPhysicalMember(1
        >>> print(members)
        """
        count = self.GetAnalyticalMemberCountForPhysicalMember(physicalMemberId)
        if count == 0:
            return []
        vt_memberList = make_safe_array_long(count)
        vt_memberList_ref = make_variant_vt_ref(vt_memberList, automation.VT_AR
        retval = self._geometry.GetAnalyticalMembersForPhysicalMember(physicalMe
        if retval <= 0:
            raise_os_error_if_error_code(retval)
        return list(vt_memberList_ref[0])
```

[docs]

```
    def GetLastPhysicalMemberNo(self):
        """
        Get the last physical member number.

        Returns
        -------
        int
            Last physical member number.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> last_no = staad_obj.Geometry.GetLastPhysicalMemberNo()
        >>> print(last_no)
        """
        retval = self._geometry.GetLastPhysicalMemberNo()
        return int(retval)
```

[docs]

```
    def GetNoOfSelectedPhysicalMembers(self):
        """
        Get the number of selected physical members.

        Returns
        -------
        int
            Number of selected physical members.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetNoOfSelectedPhysicalMembers()
        >>> print(count)
        """
        retval = self._geometry.GetNoOfSelectedPhysicalMembers()
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return int(retval)
```

[docs]

```
    def GetSelectedPhysicalMembers(self):
        """
        Get the list of selected physical members.

        Returns
        -------
        list of int
            List of selected physical member IDs.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> members = staad_obj.Geometry.GetSelectedPhysicalMembers()
        >>> print(members)
        """
        count = self.GetNoOfSelectedPhysicalMembers()
        if count == 0:
            return []
        vt_memberList = make_safe_array_long(count)
        vt_memberList_ref = make_variant_vt_ref(vt_memberList, automation.VT_AR
        self._geometry.GetSelectedPhysicalMembers(vt_memberList_ref)
        return list(vt_memberList_ref[0])
```

[docs]

```
    def GetPhysicalMemberCount(self):
        """
        Get the count of physical members in the model.
        Returns
        -------
        int
            Count of physical members.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetPhysicalMemberCount()
        >>> print(count)
        """
        retval = self._geometry.GetPhysicalMemberCount()
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return int(retval)
```

[docs]

```
    def GetPhysicalMemberList(self):
        """
        Get the list of physical members in the model.

        Returns
        -------
        list of int
            List of physical member IDs.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> members = staad_obj.Geometry.GetPhysicalMemberList()
        >>> print(members)
        """
        count = self.GetPhysicalMemberCount()
        if count == 0:
            return []
        vt_memberList = make_safe_array_long(count)
        vt_memberList_ref = make_variant_vt_ref(vt_memberList, automation.VT_AR
        self._geometry.GetPhysicalMemberList(vt_memberList_ref)
        return list(vt_memberList_ref[0])
```

[docs]

```
    def GetPhysicalMemberUniqueID(self, physicalMemberId: int):
        """
        Get the unique ID of a physical member.

        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member.

        Returns
        -------
        str
            Unique ID of the physical member.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> uid = staad_obj.Geometry.GetPhysicalMemberUniqueID(1)
        >>> print(uid)
        """
        retval = self._geometry.GetPhysicalMemberUniqueID(physicalMemberId)
        return retval
```

[docs]

```
    def GetPMemberCount(self):
        """
        Get the count of physical members in the model.

        Returns
        -------
        int
            Count of physical members.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> count = staad_obj.Geometry.GetPMemberCount()
        >>> print(count)
        """
        retval = self._geometry.GetPMemberCount()
        return int(retval)
```

[docs]

```
    def SelectMultiplePhysicalMembers(self, physicalMemberList: list):
        """
        Select multiple physical members.

        Parameters
        ----------
        physicalMemberList : list of int
            List of physical member IDs to select.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SelectMultiplePhysicalMembers([1,2,3])
        """
        vt_physicalMemberList = make_safe_array_long_input(physicalMemberList)
        self._geometry.SelectMultiplePhysicalMembers(vt_physicalMemberList)
```

[docs]

```
    def SelectPhysicalMember(self, physicalMemberId: int):
        """
        Select a physical member.

        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member to select.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SelectPhysicalMember(1)
        """
        self._geometry.SelectPhysicalMember(physicalMemberId)
```

[docs]

```
    def SetPhysicalMemberUniqueID(self, physicalMemberId: int, uniqueId: str):
        """
        Set the unique ID for a physical member.

        Parameters
        ----------
        physicalMemberId : int
            ID of the physical member.
        uniqueId : str
            Unique ID to set for the physical member.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetPhysicalMemberUniqueID(1, "physical-member-uu
        """
        self._geometry.SetPhysicalMemberUniqueID(physicalMemberId, uniqueId)
```

[docs]

```
    def SetPID(self, EntityNo: int, EntityType: int, PropertyID: int):
        """
        Set the property ID of a member.

        Parameters
        ----------
        EntityNo : int
            ID of the entity.
        EntityType : int
            Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Sol
        PropertyID : int
            Property ID to set for the member.

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetPID(1, 2, 5)
        """
        self._geometry.SetPID(EntityNo, EntityType, PropertyID)
```

[docs]

```
    def GetPID(self, EntityNo: int, EntityType: int):
        """
        Get the property ID of a member.

        Parameters
        ----------
        EntityNo : int
            ID of the entity.
        EntityType : int
            Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Sol

        Returns
        -------
        int
            Property ID of the member.
        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> pid = staad_obj.Geometry.GetPID(1)
        >>> print(pid)
        """
        retval = self._geometry.GetPID(EntityNo, EntityType)
        return int(retval)
```

[docs]

```
    def GetFlagForHiddenEntities(self):
        """
        Get the flag specified for consideration of hidden entities (nodes and p

        Returns
        -------
        int
            All entities = 0 (Default option), Ignore Hidden entities = 1, Only

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> flag = staad_obj.Geometry.GetFlagForHiddenEntities()
        >>> print(flag)
        """
        retval = self._geometry.GetFlagForHiddenEntities()
        return int(retval)
```

[docs]

```
    def GetMemberIncidence_CIS2(self, memberId: int):
        """
        Get the incidence of a member in CIS/2 format.

        Parameters
        ----------
        memberId : int
            ID of the member.

        Returns
        -------
        tuple
            (unique_str_id, start_node, end_node) where:
            unique_str_id : Unique string ID of the member. (str)
            start_node : Start node ID of the member. (int)
            end_node : End node ID of the member. (int)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> incidence = staad_obj.Geometry.GetMemberIncidence_CIS2(1)
        >>> print(incidence)
        """
        unique_str_id = make_safe_str()
        unique_str_id_ref = make_variant_vt_ref(unique_str_id, automation.VT_BS

        start_node_vt = make_safe_array_long(1)
        start_node_ref = make_variant_vt_ref(start_node_vt, automation.VT_I4)

        end_node_vt = make_safe_array_long(1)
        end_node_ref = make_variant_vt_ref(end_node_vt, automation.VT_I4)

        retval = self._geometry.GetMemberIncidence_CIS2(memberId, unique_str_id
        if retval < 0:
            raise_os_error_if_error_code(retval)
        return (unique_str_id_ref[0], start_node_ref[0], end_node_ref[0])
```

[docs]

```
    def GetNodeIncidence_CIS2(self, nodeId: int):
        """
        Get the incidence of a node in CIS/2 format.

        Parameters
        ----------
        nodeId : int
            ID of the node.

        Returns
        -------
        tuple
            (unique_str_id, x, y, z) where:
            unique_str_id : Unique string ID of the node. (str)
            x : X-coordinate of the node. (float)
            y : Y-coordinate of the node. (float)
            z : Z-coordinate of the node. (float)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> incidence = staad_obj.Geometry.GetNodeIncidence_CIS2(1)
        >>> print(incidence)
        """
        unique_str_id = make_safe_str()
        unique_str_id_ref = make_variant_vt_ref(unique_str_id, automation.VT_BS

        x_vt = make_safe_array_double(1)
        x_ref = make_variant_vt_ref(x_vt, automation.VT_R8)

        y_vt = make_safe_array_double(1)
        y_ref = make_variant_vt_ref(y_vt, automation.VT_R8)
        z_vt = make_safe_array_double(1)
        z_ref = make_variant_vt_ref(z_vt, automation.VT_R8)

        retval = self._geometry.GetNodeIncidence_CIS2(nodeId, unique_str_id_ref
        if retval <= 0:
            raise_os_error_if_error_code(retval)
        return (unique_str_id_ref[0], x_ref[0], y_ref[0], z_ref[0])
```

[docs]

```
    def GetPlateIncidence_CIS2(self, plateId: int):
        """
        Get the incidence of a plate in CIS/2 format.

        Parameters
        ----------
        plateId : int
            ID of the plate.

        Returns
        -------
        tuple
            (unique_str_id, nodeA, nodeB, nodeC, nodeD) where:
            unique_str_id : Unique string ID of the plate. (str)
            nodeA : Node A ID of the plate. (int)
            nodeB : Node B ID of the plate. (int)
            nodeC : Node C ID of the plate. (int)
            nodeD : Node D ID of the plate. (int)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> incidence = staad_obj.Geometry.GetPlateIncidence_CIS2(1)
        >>> print(incidence)
        """
        unique_str_id = make_safe_str()
        unique_str_id_ref = make_variant_vt_ref(unique_str_id, automation.VT_BS

        nodeA_vt = make_safe_array_long(1)
        nodeA_ref = make_variant_vt_ref(nodeA_vt, automation.VT_I4)

        nodeB_vt = make_safe_array_long(1)
        nodeB_ref = make_variant_vt_ref(nodeB_vt, automation.VT_I4)

        nodeC_vt = make_safe_array_long(1)
        nodeC_ref = make_variant_vt_ref(nodeC_vt, automation.VT_I4)

        nodeD_vt = make_safe_array_long(1)
        nodeD_ref = make_variant_vt_ref(nodeD_vt, automation.VT_I4)

        retval = self._geometry.GetPlateIncidence_CIS2(plateId, unique_str_id_r
        if retval <= 0:
            raise_os_error_if_error_code(retval)
        return (unique_str_id_ref[0], nodeA_ref[0], nodeB_ref[0], nodeC_ref[0],
```

[docs]

```
    def GetSolidIncidence_CIS2(self, solidId: int):
        """
        Get the incidence of a solid in CIS/2 format.

        Parameters
        ----------
        solidId : int
            ID of the solid.

        Returns
        -------
        tuple
            (unique_str_id, nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, nodeG, noc
            unique_str_id : Unique string ID of the solid. (str)
            nodeA : Node A ID of the solid. (int)
            nodeB : Node B ID of the solid. (int)
            nodeC : Node C ID of the solid. (int)
            nodeD : Node D ID of the solid. (int)
            nodeE : Node E ID of the solid. (int)
            nodeF : Node F ID of the solid. (int)
            nodeG : Node G ID of the solid. (int)
            nodeH : Node H ID of the solid. (int)

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> incidence = staad_obj.Geometry.GetSolidIncidence_CIS2(1)
        >>> print(incidence)
        """
        unique_str_id = make_safe_str()
        unique_str_id_ref = make_variant_vt_ref(unique_str_id, automation.VT_BS

        nodeA_vt = make_safe_array_long(1)
        nodeA_ref = make_variant_vt_ref(nodeA_vt, automation.VT_I4)

        nodeB_vt = make_safe_array_long(1)
        nodeB_ref = make_variant_vt_ref(nodeB_vt, automation.VT_I4)

        nodeC_vt = make_safe_array_long(1)
        nodeC_ref = make_variant_vt_ref(nodeC_vt, automation.VT_I4)

        nodeD_vt = make_safe_array_long(1)
        nodeD_ref = make_variant_vt_ref(nodeD_vt, automation.VT_I4)

        nodeE_vt = make_safe_array_long(1)
        nodeE_ref = make_variant_vt_ref(nodeE_vt, automation.VT_I4)

        nodeF_vt = make_safe_array_long(1)
        nodeF_ref = make_variant_vt_ref(nodeF_vt, automation.VT_I4)

        nodeG_vt = make_safe_array_long(1)
        nodeG_ref = make_variant_vt_ref(nodeG_vt, automation.VT_I4)

        nodeH_vt = make_safe_array_long(1)
        nodeH_ref = make_variant_vt_ref(nodeH_vt, automation.VT_I4)

        retval = self._geometry.GetSolidIncidence_CIS2(solidId, unique_str_id_re
        if retval <= 0:
            raise_os_error_if_error_code(-1)
        return (unique_str_id_ref[0], nodeA_ref[0], nodeB_ref[0], nodeC_ref[0],
```

[docs]

```
    def SetCheckForIdenticalEntity(self, entityType: int, checkFlag: bool):
        """
        This API will set whether to enable checking for existing identical enti
        If set is enabled, time taken by the corresponding add/create multiple e
        otherwise if set is disabled, time taken by corresponding APIs will be c

        Parameters
        ----------
        entityType : int
            Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Soli
        checkFlag : bool
            Flag to check for identical entities.

        Returns
        -------
        bool
            True if successful, False otherwise.

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetCheckForIdenticalEntity(1, True)
        """
        if entityType not in (1, 2, 3, 4, 5):
            raise_os_error_if_error_code(-100)
        retval = self._geometry.SetCheckForIdenticalEntity(entityType, int(checl
        return bool(retval)
```

[docs]

```
    def SetFlagForHiddenEntities(self, flag: int):
        """
        Set the flag specified for consideration of hidden entities (nodes and p

        Parameters
        ----------
        flag : int
            All entities = 0 (Default option), Ignore Hidden entities = 1, Only

        Returns
        -------
        None

        Examples
        --------
        >>> from openstaadpy import os_analytical
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.Geometry.SetFlagForHiddenEntities(1)
        """
        if flag not in (0, 1, 2):
            raise_os_error_if_error_code(-100)
        self._geometry.SetFlagForHiddenEntities(flag)
```