```
#--------------------------------------------------------------------------------
# Copyright (c) Bentley Systems, Incorporated. All rights reserved.
# See COPYRIGHT.md in the repository root for full copyright notice
#--------------------------------------------------------------------------------
from unittest import result
from winreg import REG_REFRESH_HIVE

from openstaadpy.os_analytical.oserrors import *
from .openStaadHelper import *
from comtypes import automation
from comtypes import CoInitialize
from .ossupport import OSSupport
import time
```

[docs]

```
class OSOutput:
    CoInitialize()
```

[docs]

```
    def __init__(self, staad_obj):
        self._staad = staad_obj
        self._output = self._staad.Output
        self._support = OSSupport(self._staad)
        self._functions= [
            "GetOutputUnitForDimension",
            "GetOutputUnitForSectDimension",
            "GetOutputUnitForSectArea",
            "GetOutputUnitForSectInertia",
            "GetOutputUnitForSectModulus",
            "GetOutputUnitForDensity",
            "GetOutputUnitForDisplacement",
            "GetOutputUnitForRotation",
            "GetOutputUnitForForce",
            "GetOutputUnitForMoment",
            "GetOutputUnitForDistForce",
            "GetOutputUnitForDistMoment",
            "GetOutputUnitForStress",
            "GetNodeDisplacements",
            "GetSupportReactions",
            "GetMemberEndDisplacements",
            "GetMemberEndForces",
            "GetAllPlateCenterStressesAndMoments",

            "GetPlateCenterNormalPrincipalStresses",

            "GetAllPlateCenterForces",
            "GetAllPlateCenterMoments",
            "GetAllSolidNormalStresses",
            "GetMemberSteelDesignRatio",
            "GetMinMaxBendingMoment",
            "GetMinMaxShearForce",
            "GetMinMaxAxialForce",
            "GetMaxSectionDisplacement",
            "GetMaxBeamStresses",
            "GetIntermediateMemberTransDisplacements",
            "GetAllPlateCenterPrincipalStressesAndAngles",
            "GetPlateCenterVonMisesStresses",
            "GetAllSolidShearStresses",
            "GetAllSolidPrincipalStresses",
            "GetAllSolidVonMisesStresses",
            "GetIntermediateMemberForcesAtDistance",
            "GetIntermediateDeflectionAtDistance",
            "GetPlateCornerForces",
            "GetMemberDesignSectionName",
            "AreResultsAvailable",
            "GetNLLoadStep",
            "GetNLNodeDisplacements",
            "GetIntermediateMemberAbsTransDisplacements",
            "GetNoOfModesExtracted",
            "GetModeFrequency",
            "GetModalDisplacementAtNode",
            "GetModalMassParticipationFactors",
            "GetStaticCheckResult",
            "GetMatInfluenceAreas",
            "GetBasePressures",
            "IsBucklingAnalysisResultsAvailable",
            "GetNoOfBucklingFactors",
            "GetBucklingFactor",
            "GetBucklingModeDisplacementAtNode",
            "GetResultantForceAlongLineForPlateList",
            "GetResultantForceAlongLineForParametricSurface",
            "GetPlateStressAtPoint",
            "GetTimeHistoryIntegrationStepInfo",
            "GetTimeHistoryResponseAtTime",
            "GetTimeHistoryResponse",
            "GetTimeHistoryResponseMinMax",
            "GetMemberSteelDesignResults",
            "GetMemberSteelDesignMinFailureRatio",
            "GetMemberSteelDesignMaxFailureRatio",
            "IsMultipleMemberSteelDesignResultsAvailable",
            "GetSteelDesignParameterBlockCount",
            "GetSteelDesignParameterBlockNameByIndex",
            "GetMultipleMemberSteelDesignRatio",
            "GetMultipleMemberSteelDesignResults",
            "GetMultipleMemberSteelDesignMaxRatio",
            "GetAllPlateCenterPrincipalStressesAndAnglesEx",
            "GetPMemberEndForces",
            "GetPMemberIntermediateForcesAtDistance"

        ]

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

[docs]

```
    def GetOutputUnitForDimension(self):
        """
        Get the output unit for dimension.

        Returns
        -------
        str
            The unit for dimension.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForDimension()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForDimension(refUnit)
        if not result:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForSectDimension(self):
        """
        Get the output unit for section dimension.

        Returns
        -------
        str
            The unit for section dimension.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForSectDimension()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForSectDimension(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForSectArea(self):
        """
        Get the output unit for section area.

        Returns
        -------
        str
            The unit for section area.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForSectArea()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForSectArea(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForSectInertia(self):
        """
        Get the output unit for section inertia.

        Returns
        -------
        str
            The unit for section inertia.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForSectInertia()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForSectInertia(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForSectModulus(self):
        """
        Get the output unit for section modulus.

        Returns
        -------
        str
            The unit for section modulus.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForSectModulus()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForSectModulus(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForDensity(self):
        """
        Get the output unit for density.

        Returns
        -------
        str
            The unit for density.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForDensity()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForDensity(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForDisplacement(self):
        """
        Get the output unit for displacement.

        Returns
        -------
        str
            The unit for displacement.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForDisplacement()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForDisplacement(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForRotation(self):
        """
        Get the output unit for rotation.

        Returns
        -------
        str
            The unit for rotation.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForRotation()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForRotation(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForForce(self):
        """
        Get the output unit for force.

        Returns
        -------
        str
            The unit for force.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForForce()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForForce(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForMoment(self):
        """
        Get the output unit for moment.

        Returns
        -------
        str
            The unit for moment.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForMoment()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForMoment(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForDistForce(self):
        """
        Get the output unit for distributed force.

        Returns
        -------
        str
            The unit for distributed force.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForDistForce()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForDistForce(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForDistMoment(self):
        """
        Get the output unit for distributed moment.

        Returns
        -------
        str
            The unit for distributed moment.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForDistMoment()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForDistMoment(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetOutputUnitForStress(self):
        """
        Get the output unit for stress.

        Returns
        -------
        str
            The unit for stress.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> unit = staad_obj.Output.GetOutputUnitForStress()
        """
        unit = create_bstr()
        refUnit = make_byref(unit)
        result = self._output.GetOutputUnitForStress(refUnit)
        if result < 0:
            raise_os_error_if_error_code(-1)
        return unit.value
```

[docs]

```
    def GetNodeDisplacements(self, nodeNo:int, loadCaseNo:int):
        """
        Get the displacements for a given node.

        Parameters
        ----------
        nodeNo : int
            The node number.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list
            List with nodal translational displacements in X, Y and Z directions

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> nodeList = staad_obj.Geometry.GetNodeList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 1, 1)
        >>>
        >>> displacements = staad_obj.Output.GetNodeDisplacements(nodeList[3], l
        """
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement,  automation.VT_AR
        result = self._output.GetNodeDisplacements(nodeNo, loadCaseNo, re_Displa
        if not result:
            raise_os_error_if_error_code(-1)
        displacements = list(re_Displacement[0])
        return displacements
```

[docs]

```
    def GetSupportReactions(self, nodeNo:int, loadCaseNo:int):
        """
        Get the support reactions for a given support.

        Parameters
        ----------
        nodeNo : int
            The node number.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list
            List with Reaction in GLOBAL direction:[ FX, FY, FZ, MX, MY, MZ]

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> nodeList = staad_obj.Geometry.GetNodeList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> supportReactions = staad_obj.Output.GetSupportReactions(nodeList[3],
        """
        vt_SupportReactions = make_safe_array_double(6)
        re_SupportReactions = make_variant_vt_ref(vt_SupportReactions,  automat
        result = self._output.GetSupportReactions(nodeNo, loadCaseNo, re_Suppor
        if not result:
            raise_os_error_if_error_code(-1)
        SupportReactions = list(vt_SupportReactions[0])
        return SupportReactions
```

[docs]

```
    def GetMemberEndDisplacements(self, memberNo:int, end:int, loadCaseNo:int):
        """
        Get the end displacements for a given member.

        Parameters
        ----------
        memberNo : int
            The member number.
        end : int
            The end number (0 for starting and 1 for ending).
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list
            List with end displacements of Member End in terms of X, Y, Z (in or

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> memberEndDisplacements = staad_obj.Output.GetMemberEndDisplacements(
        """
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement,  automation.VT_AI
        result = self._output.GetMemberEndDisplacements(memberNo, end, loadCaseN
        if not result:
            raise_os_error_if_error_code(-1)
        displacements = list(re_Displacement[0])
        return displacements
```

[docs]

```
    def GetMemberEndForces(self, memberNo:int, end:int, loadCaseNo:int, LocalOr(
        """
        Get the end forces for a given member.

        Parameters
        ----------
        memberNo : int
            The member number.
        end: int
            The end number (0 for starting and 1 for ending).
        loadCaseNo : int
            The load case number.
        LocalOrGlobal : int
            Local Or Global direction (0 for Local and 1 for Global).

        Returns
        -------
        list
            list with end force values FX, FY, FZ, MX, MY and MZ (in order).
        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> memberEndForces = staad_obj.Output.GetMemberEndForces(beamList[0], 0
        """
        vt_Forces = make_safe_array_double(6)
        re_Forces = make_variant_vt_ref(vt_Forces,  automation.VT_ARRAY | automa
        result = self._output.GetMemberEndForces(memberNo, end, loadCaseNo, re_F
        if not result:
            raise_os_error_if_error_code(-1)
        forces = list(re_Forces[0])
        return forces
```

[docs]

```
    def GetAllPlateCenterStressesAndMoments(self, plateNo:int, loadCaseNo:int):
        """
        Gets plate center stresses and moments for the specified plate for speci

        Parameters
        ----------
        plateNo : int
            Plate number.
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        list of float
            list of plate center stresses and moments organized in following ord
                +---------------+---------------+-------------------------------
                | List Index    | Variable      | Load Type
                +===============+===============+===============================
                | 0             | SQX           |  Shear stress on the local X f
                +---------------+---------------+-------------------------------
                | 1             | SQY           |  Shear stress on the local Y f
                +---------------+---------------+-------------------------------
                | 2             | MX            |  Moment per unit width about t
                +---------------+---------------+-------------------------------
                | 3             | MY            |  Moment per unit width about t
                +---------------+---------------+-------------------------------
                | 4             | MXY           |  Torsional Moment per unit wid
                +---------------+---------------+-------------------------------
                | 5             | SX            |  Axial stress in the local X d
                +---------------+---------------+-------------------------------
                | 6             | SY            |  Axial stress in the local Y d
                +---------------+---------------+-------------------------------
                | 7             | SXY          |  Shear stress in the local XY
                +--------------+--------------+-------------------------------
            Note :
                - For additional information, please refer to Section: "Sign Cor

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plateList = staad_obj.Geometry.GetPlateList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> plateCenterStressesAndMoments = staad_obj.Output.GetAllPlateCenterSt
        """
        vt_ForcesOrMoments = make_safe_array_double(7)
        re_ForcesOrMoments = make_variant_vt_ref(vt_ForcesOrMoments,  automatio
        result = self._output.GetAllPlateCenterStressesAndMoments(plateNo, load
        if not result:
            raise_os_error_if_error_code(-1)
        forcesOrMoments = list(re_ForcesOrMoments[0])
        return forcesOrMoments
```

[docs]

```
def GetPlateCenterNormalPrincipalStresses(self, plateNo:int, loadCaseNo:int
    """
    Get principal stresses of specified plate.

    Parameters
    ----------
    plateNo : int
        The plate number.
    loadCaseNo : int
        The load case number.

    Returns
    -------
    Tuple
        Tuple containing maximum in-plane Principal Stress at Top surface of
        maximum in-plane Principal Stress at Bottom surface of plate and mir

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>>
    >>> beamList = staad_obj.Geometry.GetPlateList()
    >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
```

```
        >>> plateCenterNormalPrincipalStresses = staad_obj.Output.GetPlateCenter
        """
        safe_pdSMAXTop = make_safe_array_double(1)
        safe_pdSMINTop = make_safe_array_double(1)
        safe_pdSMAXBottom = make_safe_array_double(1)
        safe_pdSMINBottom = make_safe_array_double(1)
        pdSMAXTop = make_variant_vt_ref(safe_pdSMAXTop,  automation.VT_R8)
        pdSMINTop = make_variant_vt_ref(safe_pdSMINTop,  automation.VT_R8)
        pdSMAXBottom = make_variant_vt_ref(safe_pdSMAXBottom,  automation.VT_R8
        pdSMINBottom = make_variant_vt_ref(safe_pdSMINBottom,  automation.VT_R8
        self._output.GetPlateCenterNormalPrincipalStresses(plateNo, loadCaseNo,
        return (pdSMAXTop[0], pdSMINTop[0], pdSMAXBottom[0], pdSMINBottom[0])
```

[docs]

```
    def GetAllPlateCenterForces(self, plateNo:int, loadCaseNo:int):
        """
        Get the plate center stresses (Shear & Membrane) for the specified plate

        Parameters
        ----------
        plateNo : int
            The plate number.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list of float
            list of plate center forces organized in following order:
                +---------------+---------------+-------------------------------
                | List Index    | Variable      | Load Type
                +===============+===============+===============================
                | 0             | SQX           |  Shear stress on the local X t
                +---------------+---------------+-------------------------------
                | 1             | SQY           |  Shear stress on the local Y t
                +---------------+---------------+-------------------------------
                | 2             | SX            |  Axial stress in the local X d
                +---------------+---------------+-------------------------------
                | 3             | SY            |  Axial stress in the local Y d
                +---------------+---------------+-------------------------------
                | 4             | SXY           |  Shear stress in the local XY
                +---------------+---------------+-------------------------------
            Note :
                - For additional information, please refer to Section: "Sign Co


        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plateList = staad_obj.Geometry.GetPlateList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> plateCenterForces = staad_obj.Output.GetAllPlateCenterForces(plateLi
        """
        vt_Forces = make_safe_array_double(5)
        re_Forces = make_variant_vt_ref(vt_Forces,  automation.VT_ARRAY | automa
        result = self._output.GetAllPlateCenterForces(plateNo, loadCaseNo, re_Fo
        if not result:
            raise_os_error_if_error_code(-1)
        forces = list(re_Forces[0])
        return forces
```

[docs]

```
    def GetAllPlateCenterMoments(self, plateNo:int, loadCaseNo:int):
        """
        Get the plate center stresses (Shear & Membrane) for the specified plate

        Parameters
        ----------
        plateNo : int
            The plate number.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list of float
            list of plate center moments organized in following order:
                +---------------+---------------+-------------------------------
                | List Index    | Variable      | Load Type
                +===============+===============+===============================
                | 0             | MX            |  Moment per unit width about t
                +---------------+---------------+-------------------------------
                | 1             | MY            |  Moment per unit width about t
                +---------------+---------------+-------------------------------
                | 2             | MXY           |  Torsional Moment per unit wid
                +---------------+---------------+-------------------------------
            Note :
                - For additional information, please refer to Section: "Sign Con


        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plateList = staad_obj.Geometry.GetPlateList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> plateCenterMoments = staad_obj.Output.GetAllPlateCenterMoments(plate
        """
        vt_Moments = make_safe_array_double(3)
        re_Moments = make_variant_vt_ref(vt_Moments,  automation.VT_ARRAY | auto
        result = self._output.GetAllPlateCenterMoments(plateNo, loadCaseNo, re_M
        if not result:
            raise_os_error_if_error_code(-1)
        moments = list(re_Moments[0])
        return moments
```

[docs]

```
    def GetAllSolidNormalStresses(self, nSolidNo:int, nCorner:int, loadCaseNo:in
        """
        Gets all solid normal stresses.

        Parameters
        ----------
        nSolidNo : int
            Solid number ID.
        nCorner : int
            Corner of the solid.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        list of float
            list with solid normal stresses SXX, SYY and SZZ (in order).

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> solidList = staad_obj.Geometry.GetSolidList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> solidNormalStresses = staad_obj.Output.GetAllSolidNormalStresses(sol
        """
        vt_SolidNormalStresses = make_safe_array_double(3)
        re_SolidNormalStresses = make_variant_vt_ref(vt_SolidNormalStresses,  au
        result = self._output.GetAllSolidNormalStresses(nSolidNo, nCorner, loadC
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_SolidNormalStresses[0])
```

[docs]

```
    def GetMemberSteelDesignRatio(self, beamNo:int):
        """
        Gets the critical steel design ratio for a steel member. This method wi

        Parameters
        ----------
        beamNo : int
            The beam number ID.

        Returns
        -------
        float
            Returns the critical steel design ratio.
            Returns -999 if analysis is performed but the member is not designed
            Returns -1 if analysis is not performed.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> memberSteelDesignRatio = staad_obj.Output.GetMemberSteelDesignRatio(
        """
        safe_MemberSteelDesignRatio = make_safe_array_double(1)
        re_MemberSteelDesignRatio = make_variant_vt_ref(safe_MemberSteelDesignRa
        result = self._output.GetMemberSteelDesignRatio(beamNo, re_MemberSteelDe
        if not result:
            raise_os_error_if_error_code(-1)
        return re_MemberSteelDesignRatio[0]
```

[docs]

```
    def GetMinMaxBendingMoment(self, memberNo:int, dir:str, loadCaseNo:int):
        """
        Gets the maximum and minimum bending moments and their locations for spe

        Parameters
        ----------
        memberNo : int
            Member number ID.
        dir: str
            Bending direction in LOCAL coordinate: MY or MZ.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        tuple of float
            tuple with minimum bending moment, the location along the length of
            maximum bending moment and the location along the length of the memb

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> minMaxBendingMoment = staad_obj.Output.GetMinMaxBendingMoment(beamLi
        """
        safe_dMin =  make_safe_array_double(1)
        safe_dMinPos =  make_safe_array_double(1)
        safe_dMax =  make_safe_array_double(1)
        safe_dMaxPos =  make_safe_array_double(1)
        dMin = make_variant_vt_ref(safe_dMin,  automation.VT_R8)
        dMinPos = make_variant_vt_ref(safe_dMinPos,  automation.VT_R8)
        dMax = make_variant_vt_ref(safe_dMax,  automation.VT_R8)
        dMaxPos = make_variant_vt_ref(safe_dMaxPos,  automation.VT_R8)
        result = self._output.GetMinMaxBendingMoment(memberNo, dir, loadCaseNo,
        if not result:
            raise_os_error_if_error_code(-1)
        return (dMin[0], dMinPos[0], dMax[0], dMaxPos[0])
```

[docs]

```
    def GetMinMaxShearForce(self, memberNo:int, dir:str, loadCaseNo:int):
        """
        Gets the maximum and minimum shear forces and their locations for specif

        Parameters
        ----------
        memberNo : int
            Member number ID.
        dir: str
            Force direction in LOCAL coordinate: FY or FZ.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        tuple of float
            tuple with minimum bending shear, the location along the length of t
            maximum bending shear and the location along the length of the membe

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> minMaxShearForce = staad_obj.Output.GetMinMaxShearForce(beamList[0],
        """
        safe_dMin = make_safe_array_double(1)
        safe_dMinPos = make_safe_array_double(1)
        safe_dMax = make_safe_array_double(1)
        safe_dMaxPos = make_safe_array_double(1)
        dMin = make_variant_vt_ref(safe_dMin,  automation.VT_R8)
        dMinPos = make_variant_vt_ref(safe_dMinPos,  automation.VT_R8)
        dMax = make_variant_vt_ref(safe_dMax,  automation.VT_R8)
        dMaxPos = make_variant_vt_ref(safe_dMaxPos,  automation.VT_R8)
        result = self._output.GetMinMaxShearForce(memberNo, dir, loadCaseNo, dM
        if not result:
            raise_os_error_if_error_code(-1)
        return (dMin[0], dMinPos[0], dMax[0], dMaxPos[0])
```

[docs]

```
    def GetMinMaxAxialForce(self, memberNo:int, loadCaseNo:int):
        """
        Gets the maximum and minimum axial forces and their locations for specif

        Parameters
        ----------
        memberNo : int
            Member number ID.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        tuple of float
            tuple with minimum axial force, the location along the length of the
            maximum axial force and the location along the length of the member

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> minMaxAxialForce = staad_obj.Output.GetMinMaxAxialForce(beamList[0],
        """
        safe_dMin = make_safe_array_double(1)
        safe_dMinPos = make_safe_array_double(1)
        safe_dMax = make_safe_array_double(1)
        safe_dMaxPos = make_safe_array_double(1)
        dMin = make_variant_vt_ref(safe_dMin,  automation.VT_R8)
        dMinPos = make_variant_vt_ref(safe_dMinPos,  automation.VT_R8)
        dMax = make_variant_vt_ref(safe_dMax,  automation.VT_R8)
        dMaxPos = make_variant_vt_ref(safe_dMaxPos,  automation.VT_R8)
        result = self._output.GetMinMaxAxialForce(memberNo, loadCaseNo, dMin, dM
        if not result:
            raise_os_error_if_error_code(-1)
        return (dMin[0], dMinPos[0], dMax[0], dMaxPos[0])
```

[docs]

```
    def GetMaxSectionDisplacement(self, memberNo:int, dir:str, loadCaseNo:int):
        """
        Gets the maximum section displacements for specified member number, dire

        Parameters
        ----------
        memberNo : int
            Member number ID.
        dir: str
            Direction in GLOBAL: X, Y or Z
        loadCaseNo : int
            The load case number.

        Returns
        -------
        tuple of float
            tuple with maximum section displacement in specified direction and t

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> maxSectionDisplacement = staad_obj.Output.GetMaxSectionDisplacement(
        """
        safe_dMax = make_safe_array_double(1)
        safe_dMaxPos = make_safe_array_double(1)
        dMax = make_variant_vt_ref(safe_dMax,  automation.VT_R8)
        dMaxPos = make_variant_vt_ref(safe_dMaxPos,  automation.VT_R8)
        result = self._output.GetMaxSectionDisplacement(memberNo, dir, loadCaseN
        if not result:
            raise_os_error_if_error_code(-1)
        return dMax[0], dMaxPos[0]
```

[docs]

```
    def GetMaxBeamStresses(self, beamNo:int, loadCaseNo:int):
        """
        Gets the maximum beam Stresses for Beam.
        Parameters
        ----------
        beamNo : int
            Beam number ID.
        loadCaseNo : int
            The load case number.

        Returns
        -------
        tuple
            tuple with value of maximum compressive stress, integer value (0 for
            value of maximum tensile stress and integer value (0 for non-corner-

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> maxBeamStresses = staad_obj.Output.GetMaxBeamStresses(beamList[0], l
        """
        safe_pdCompStress = make_safe_array_double(1)
        safe_nCompCorner = make_safe_array_long(1)
        safe_pdTensileStress = make_safe_array_double(1)
        safe_nTensileCorner = make_safe_array_long(1)
        pdCompStress = make_variant_vt_ref(safe_pdCompStress,  automation.VT_R8)
        nCompCorner = make_variant_vt_ref(safe_nCompCorner,  automation.VT_I4)
        pdTensileStress = make_variant_vt_ref(safe_pdTensileStress,  automation
        nTensileCorner = make_variant_vt_ref(safe_nTensileCorner,  automation.V
        result = self._output.GetMaxBeamStresses(beamNo, loadCaseNo, pdCompStres
        if result < 0:
            raise_os_error_if_error_code(result)
        return (pdCompStress[0], nCompCorner[0], pdTensileStress[0], nTensileCo
```

[docs]

```
    def GetIntermediateMemberTransDisplacements(self, memberNo: int, distance: 
        """
        Get section displacement (or relative displacements) of a beam section 

        Parameters
        ----------
        memberNo : int
            Member number ID.
        distance : int
            Distance from starting end in terms of member length.
        loadCaseNo : int
            Load Case reference ID.
        Returns
        -------
        list
            List of relative displacements at specified section in terms of LOCA

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> intermediateMemberTransDisplacements = staad_obj.Output.GetIntermedi
        """
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement, automation.VT_ARI
        result = self._output.GetIntermediateMemberTransDisplacements(memberNo,
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Displacement[0])
```

[docs]

```
    def GetAllPlateCenterPrincipalStressesAndAngles(self, plateNo: int, loadCase
        """
        Get all plate center principal stresses and angles.

        Parameters
        ----------
        plateNo : int
            Plate number ID.
        loadCaseNo : int
            Case reference ID.

        Returns
        -------
        list of float
            List of float values orgainized according to below table:
                +---------------+------------------------------------------------
                | Variable      | Description
                +===============+================================================
                | pdStresses[0] | Top-Maximum in-plane Principal stress
                +---------------+------------------------------------------------
                | pdStresses[1] | Top-Minimum in-plane Principal stress
                +---------------+------------------------------------------------
                | pdStresses[2] | Top-Maximum in-plane Shear stress
                +---------------+------------------------------------------------
                | pdStresses[3] | Bottom-Maximum in-plane Principal stress
                +---------------+------------------------------------------------
                | pdStresses[4] | Bottom-Minimum in-plane Principal stress
                +---------------+------------------------------------------------
                | pdStresses[5] | Bottom-Maximum in-plane Shear stress
                +---------------+------------------------------------------------
                | pdStresses[6] | Top-Angle which determines direction of maximu
                +---------------+------------------------------------------------
                | pdStresses[7] | Bottom-Angle which determines direction of max
                +---------------+------------------------------------------------


        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plateList = staad_obj.Geometry.GetPlateList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> plateCenterPrincipalStressesAndAngles = staad_obj.Output.GetAllPlate
        """
        vt_Values = make_safe_array_double(8)
        re_Values = make_variant_vt_ref(vt_Values, automation.VT_ARRAY | automa
        result = self._output.GetAllPlateCenterPrincipalStressesAndAngles(plateN
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Values[0])
```

[docs]

```
    def GetPlateCenterVonMisesStresses(self, plateNo: int, loadCaseNo: int):
        """
        Gets Von Mises stresses at center of specified plate for specified load

        Parameters
        ----------
        plateNo : int
            Plate number ID.
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        tuple
            Tuple of Von Mises stress on the top surface of the plate and Von Mi

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plateList = staad_obj.Geometry.GetPlateList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> vonMisesStresses = staad_obj.Output.GetPlateCenterVonMisesStresses(p
        """
        safe_pdVONT = make_safe_array_double(1)
        safe_pdVONB = make_safe_array_double(1)
        re_pdVONT = make_variant_vt_ref(safe_pdVONT, automation.VT_R8)
        re_pdVONB = make_variant_vt_ref(safe_pdVONB, automation.VT_R8)
        result = self._output.GetPlateCenterVonMisesStresses(plateNo, loadCaseN
        if not result:
            raise_os_error_if_error_code(-1)
        return (re_pdVONT[0], re_pdVONB[0])
```

[docs]

```
    def GetAllSolidShearStresses(self, nSolidNo: int, nCorner: int, loadCaseNo:
        """
        Get all solid shear stresses.

        Parameters
        ----------
        nSolidNo : int
            Solid number ID.
        nCorner : int
            Corner of the solid.
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        list of float
            List of shear stresses SXY, SYZ, SZX in same order.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> solidList = staad_obj.Geometry.GetSolidList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> shearStresses = staad_obj.Output.GetAllSolidShearStresses(solidList[
        """
        vt_Values = make_safe_array_double(3)
        re_Values = make_variant_vt_ref(vt_Values, automation.VT_ARRAY | automa
        result = self._output.GetAllSolidShearStresses(nSolidNo, nCorner, loadC
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Values[0])
```

[docs]

```
def GetAllSolidPrincipalStresses(self, nSolidNo: int, nCorner: int, loadCase
    """
    Get all solid principal stresses.

    Parameters
    ----------
    nSolidNo : int
        Solid number ID.
    nCorner : int
        Corner of the solid.
    loadCaseNo : int
        Load Case reference ID.

    Returns
    -------
    list of float
        List of principal stresses S_1, S_2, S_3 in same order.

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>>
    >>> solidList = staad_obj.Geometry.GetSolidList()
    >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> principalStresses = staad_obj.Output.GetAllSolidPrincipalStresses(so
    """
    vt_Values = make_safe_array_double(3)
    re_Values = make_variant_vt_ref(vt_Values, automation.VT_ARRAY | automa
    result = self._output.GetAllSolidPrincipalStresses(nSolidNo, nCorner, lo
    if not result:
        raise_os_error_if_error_code(-1)
    return list(re_Values[0])
```

[docs]

```
def GetAllSolidVonMisesStresses(self, nSolidNo: int, nCorner: int, loadCaseN
    """
    Get all solid Von Mises stresses.

    Parameters
    ----------
    nSolidNo : int
        Solid number ID.
    nCorner : int
        Corner of the solid.
    loadCaseNo : int
        Load Case reference ID.
    Returns
    -------
    list of float
        List of Von Mises stresses.

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>>
    >>> solidList = staad_obj.Geometry.GetSolidList()
    >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> vonMisesStresses = staad_obj.Output.GetAllSolidVonMisesStresses(soli
    """
    vt_Values = make_safe_array_double(1)
    re_Values = make_variant_vt_ref(vt_Values, automation.VT_R8)
    result = self._output.GetAllSolidVonMisesStresses(nSolidNo, nCorner, lo
    if not result:
        raise_os_error_if_error_code(-1)
    return re_Values[0]
```

[docs]

```
    def GetIntermediateMemberForcesAtDistance(self, memberNo: int, distance: fl
        """
        Gets sectional forces and moments for specified member number, distance,

        Parameters
        ----------
        memberNo : int
            Member number ID.
        distance : float
            Distance from the starting end of the member.
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        list of float
            List of Section axial force, Shear force in LOCAL Y & Z direction, T

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beamList = staad_obj.Geometry.GetBeamList()
        >>> loadCases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> memberForces = staad_obj.Output.GetIntermediateMemberForcesAtDistanc
        """
        vt_Forces = make_safe_array_double(6)
        re_Forces = make_variant_vt_ref(vt_Forces, automation.VT_ARRAY | automat
        result = self._output.GetIntermediateMemberForcesAtDistance(memberNo, d
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Forces[0])
```

[docs]

```
def GetIntermediateDeflectionAtDistance(self, memberNo: int, distance: int,
    """
    Get the intermediate section deflections for specified member number and

    Parameters
    ----------
    memberNo : int
        Member number ID.
    distance : float
        Distance from the starting end of the member.
    loadCaseNo : int
        Load Case reference ID.

    Returns
    -------
    tuple
        Tuple of displacement in Y & Z direction respectively.

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>>
    >>> beam_list = staad_obj.Geometry.GetBeamList()
    >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> deflections = staad_obj.Output.GetIntermediateDeflectionAtDistance(b
    """
    safe_DeflectionY = make_safe_array_double(1)
    safe_DeflectionZ = make_safe_array_double(1)
    re_DeflectionY = make_variant_vt_ref(safe_DeflectionY, automation.VT_R8)
    re_DeflectionZ = make_variant_vt_ref(safe_DeflectionZ, automation.VT_R8
    result = self._output.GetIntermediateDeflectionAtDistance(memberNo, dis
    if not result:
        raise_os_error_if_error_code(-1)
    return (re_DeflectionY[0], re_DeflectionZ[0])
```

[docs]

```
    def GetPlateCornerForces(self, plateNo: int, cornerCode:int, loadCaseNo: in
        """
        Get nodal forces at 4 corners of specified plate at load case.

        Parameters
        ----------
        plateNo : int
            Plate number ID.
        cornerCode : int
            Corner Node No.
        loadCaseNo : int
            Load case number.

        Returns
        -------
        list
            List of nodal forces at 4 corners of specified plate at load case.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> plate_list = staad_obj.Geometry.GetPlateList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> forces = staad_obj.Output.GetPlateCornerForces(plate_list[0], 1, loa
        """
        vt_Forces = make_safe_array_double(6)
        re_Forces = make_variant_vt_ref(vt_Forces, automation.VT_ARRAY | automa
        result = self._output.GetPlateCornerForces(plateNo, cornerCode, loadCas
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Forces[0])
```

[docs]

```
    def GetMemberDesignSectionName(self, beamNo: int):
        """
        Get the design section name for specified member.

        Parameters
        ----------
        beamNo : int
            Beam number ID.

        Returns
        -------
        str
            Returns design section name for the specified member. Returns empty

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> beam_list = staad_obj.Geometry.GetBeamList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> design_section_name = staad_obj.Output.GetMemberDesignSectionName(be
        """
        result = str(self._output.GetMemberDesignSectionName(beamNo))
        if result == "":
            raise_os_error_if_error_code(-1)
        return result
```

[docs]

```
    def AreResultsAvailable(self):
        """
        Check if analysis results are available or not.

        Returns
        -------
        bool
            True if results are available, False otherwise.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> are_results_available = staad_obj.Output.AreResultsAvailable()
        """
        return bool(self._output.AreResultsAvailable())
```

[docs]

```
    def GetNLLoadStep(self, loadCaseNo: int):
        """
        Gets the Load Step value used for nonlinear analysis for specified Load

        Parameters
        ----------
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        int
            Returns Load Step value.
            Returns -1 General error.
            Returns -8002 Load Case nLC not found.
            Returns -9911 Nonlinear result set is not available.


        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> load_step_value = staad_obj.Output.GetNLLoadStep(load_cases[0])
        """
        return int(self._output.GetNLLoadStep(loadCaseNo))
```

[docs]

```
    def GetNLNodeDisplacements(self, nodeNo: int, loadCaseNo: int, loadStep: in
        """
        Get the Load Level value and nodal displacements for specified node, Loa

        Parameters
        ----------
        nodeNo : int
            Node number ID.
        loadCaseNo : int
            Load Case reference ID.
        loadStep : int
            Load step number.

        Returns
        -------
        tuple
            Tuple containing value of load level and list of nodal displacements

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> nl_node_displacement = staad_obj.Output.GetNLNodeDisplacements(node_
        """
        ref_load_level = make_variant_vt_ref(make_safe_array_double(1), automat
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement, automation.VT_AR
        result = self._output.GetNLNodeDisplacements(nodeNo, loadCaseNo, loadSt
        if not result:
            raise_os_error_if_error_code(-1)
        return (ref_load_level[0], list(re_Displacement[0]))
```

[docs]

```
    def GetIntermediateMemberAbsTransDisplacements(self, memberNo: int, distanc
        """
        Gets section displacement (or relative displacements) of a beam section

        Parameters
        ----------
        memberNo : int
            Member number ID.
        distance : float
            Distance from starting end in terms of member length.
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        list of float
            List of relative displacements at specified section in terms of LOC

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> beam_list = staad_obj.Geometry.GetBeamList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> intermediate_member_abs_trans_displacements = staad_obj.Output.GetI
        """
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement, automation.VT_AR
        result = self._output.GetIntermediateMemberAbsTransDisplacements(member
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Displacement[0])
```

[docs]

```
    def GetNoOfModesExtracted(self):
        """
        Gets the number of modes extracted by a dynamic analysis.

        Returns
        -------
        int
            Number of modes extracted by a dynamic analysis.
        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> no_of_modes_extracted = staad_obj.Output.GetNoOfModesExtracted()
        """
        return self._output.GetNoOfModesExtracted()
```

[docs]

```
    def GetModeFrequency(self, modeNo: int):
        """
        Get the natural frequency (Hz) for a specified mode.

        Parameters
        ----------
        modeNo : int
            The mode number.

        Returns
        -------
        float
            Natural Frequency value for a specified mode.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> mode_frequency = staad_obj.Output.GetModeFrequency(4)
        """
        safe_freq = make_safe_array_double(1)
        ref_freq = make_variant_vt_ref(safe_freq, automation.VT_R8)
        result = self._output.GetModeFrequency(modeNo, ref_freq)
        if not result:
            raise_os_error_if_error_code(-1)
        return float(ref_freq[0])
```

[docs]

```
    def GetModalDisplacementAtNode(self, modeNo: int, nodeNo: int):
        """
        Gets the modal displacement at a specified node number and mode.

        Parameters
        ----------
        modeNo : int
```

```
            Mode number.
        nodeNo : int
            Node number ID.

        Returns
        -------
        list
            List of nodal displacements.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> mode_displacement_at_node = staad_obj.Output.GetModalDisplacementAtN
        """
        vt_Displacement = make_safe_array_double(6)
        re_Displacement = make_variant_vt_ref(vt_Displacement, automation.VT_ARI
        result = self._output.GetModalDisplacementAtNode(modeNo, nodeNo, re_Disp
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Displacement[0])
```

[docs]

```
    def GetModalMassParticipationFactors(self, modeNo: int):
        """
        Gets the modal participation factors for a specified mode number.

        Parameters
        ----------
        modeNo : int
            Mode number.

        Returns
        -------
        tuple
            Tuple of modal mass participation factor for X, Y & Z direction resp

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> mode_mass_participation_factors = staad_obj.Output.GetModalMassParti
        """
        safe_pat_x = make_safe_array_double(1)
        safe_pat_y = make_safe_array_double(1)
        safe_pat_z = make_safe_array_double(1)
        re_pat_x = make_variant_vt_ref(safe_pat_x, automation.VT_R8)
        re_pat_y = make_variant_vt_ref(safe_pat_y, automation.VT_R8)
        re_pat_z = make_variant_vt_ref(safe_pat_z, automation.VT_R8)
        result = self._output.GetModalMassParticipationFactors(modeNo, re_pat_x
        if not result:
            raise_os_error_if_error_code(-1)
        return (float(re_pat_x[0]), float(re_pat_y[0]), float(re_pat_z[0]))
```

[docs]

```
    def GetStaticCheckResult(self, loadCaseNo: int):
        """
        Gets the statics check result containing loads and reactions for specif

        Parameters
        ----------
        loadCaseNo : int
            Load Case reference ID.

        Returns
        -------
        tuple of list
            Tuple of list of loads (FX, FY, FZ, MZ, MY, MZ (in same order)) and

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> static_check_result = staad_obj.Output.GetStaticCheckResult(load_cas
        """
        safe_loads = make_safe_array_double(6)
        safe_reaction = make_safe_array_double(6)
        ref_loads = make_variant_vt_ref(safe_loads, automation.VT_ARRAY | automa
        ref_reaction = make_variant_vt_ref(safe_reaction, automation.VT_ARRAY |
        result = self._output.GetStaticCheckResult(loadCaseNo, ref_loads, ref_re
        if not result:
            raise_os_error_if_error_code(-1)
        return (list(ref_loads[0]), list(ref_reaction[0]))
```

[docs]

```
    def GetMatInfluenceAreas(self, nodelist: list):
        """
        Gets the mat influence areas for nodes supported using ELASTIC MAT comma

        Parameters
        ----------
        nodelist : list
```

```
            List of node numbers.

        Returns
        -------
        tuple of list
            Tuple of lists containing influence areas in YZ, ZX & XY place order

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> mat_influence_areas = staad_obj.Output.GetMatInfluenceAreas(node_lis
        """
        count = self._support.GetSupportCount()
        safe_yz_areas = make_safe_array_double(count)
        safe_zx_areas = make_safe_array_double(count)
        safe_xy_areas = make_safe_array_double(count)
        ref_yz_areas = make_variant_vt_ref(safe_yz_areas, automation.VT_ARRAY |
        ref_zx_areas = make_variant_vt_ref(safe_zx_areas, automation.VT_ARRAY |
        ref_xy_areas = make_variant_vt_ref(safe_xy_areas, automation.VT_ARRAY |
        safe_NodeList = make_safe_array_long_input(nodelist)
        vt_NodeList = make_variant_vt_ref(safe_NodeList, automation.VT_ARRAY | a

        result = self._output.GetMatInfluenceAreas(vt_NodeList, ref_yz_areas, re
        if not result:
            raise_os_error_if_error_code(-1)
        return (list(ref_yz_areas[0]), list(ref_zx_areas[0]), list(ref_xy_areas
```

[docs]

```
    def GetBasePressures(self, loadCaseNo: int, nodelist: list):
        """
        Gets base presure in X, Y and Z direction using Base Presure command.

        Parameters
        ----------
        loadCaseNo : int
            Load Case reference ID.
        nodelist : list
            List of node numbers.

        Returns
        -------
        tuple of list
            Tuple of list of base pressures in X, Y and Z direction per load cas

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> base_pressures_x, base_pressures_y, base_pressures_z = staad_obj.Out
        """
        count = self._support.GetSupportCount()
        safe_x_base_pressure = make_safe_array_double(count)
        safe_y_base_pressure = make_safe_array_double(count)
        safe_z_base_pressure = make_safe_array_double(count)
        ref_x_base_pressure = make_variant_vt_ref(safe_x_base_pressure, automat
        ref_y_base_pressure = make_variant_vt_ref(safe_y_base_pressure, automat
        ref_z_base_pressure = make_variant_vt_ref(safe_z_base_pressure, automat
        safe_NodeList = make_safe_array_long_input(nodelist)
        vt_NodeList = make_variant_vt_ref(safe_NodeList, automation.VT_ARRAY |
        result = self._output.GetBasePressures(loadCaseNo, vt_NodeList, ref_x_b
        if not result:
            raise_os_error_if_error_code(-1)
        return (list(ref_x_base_pressure[0]), list(ref_y_base_pressure[0]), lis
```

[docs]

```
    def IsBucklingAnalysisResultsAvailable(self):
        """
        Determines whether buckling results are available

        Returns
        -------
        bool
            True if buckling analysis results are available, False otherwise.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> buckling_analysis_results_available = staad_obj.Output.IsBucklingAna
        """
        return bool(self._output.IsBucklingAnalysisResultsAvailable())
```

[docs]

```
    def GetNoOfBucklingFactors(self):
        """
        Gets the number of buckling factors computed.

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

```
            The number of buckling factor(s) extracted by the eigen analysis.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> number_of_buckling_factors = staad_obj.Output.GetNoOfBucklingFactors
        """
        return self._output.GetNoOfBucklingFactors()
```

[docs]

```
    def GetBucklingFactor(self, buckling_mode_no: int):
        """
        Gets the buckling factor for a specified buckling mode.

        Parameters
        ----------
        buckling_mode_no : int
            Buckling mode number

        Returns
        -------
        float
            Buckling factor.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> buckling_factor = staad_obj.Output.GetBucklingFactor(1)
        """
        vt_lambda = create_variant_float(0)
        ref_lambda = make_variant_vt_ref(vt_lambda, automation.VT_R8)
        result = self._output.GetBucklingFactor(buckling_mode_no, ref_lambda)
        if not result:
            raise_os_error_if_error_code(-1)
        return ref_lambda[0]
```

[docs]

```
    def GetBucklingModeDisplacementAtNode(self, buckling_mode_no: int, node_no:
        """
        Gets the modal displacement at a specified node number and mode.

        Parameters
        ----------
        buckling_mode_no : int
            Buckling mode number.
        node_no : int
            Node number at which buckling analysis result is to be extracted.

        Returns
        -------
        list of float
            List of buckling mode displacements.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> buckling_mode_displacement_at_node = staad_obj.Output.GetBucklingMoc
        """
        disp = make_safe_array_double(6)
        ref_disp = make_variant_vt_ref(disp, automation.VT_ARRAY | automation.V
        result = self._output.GetBucklingModeDisplacementAtNode(buckling_mode_nc
        if not result:
            raise_os_error_if_error_code(-1)
        return list(ref_disp[0])
```

[docs]

```
    def GetResultantForceAlongLineForPlateList(self, plateList : list, nplates
        """
        Gets forces and moments along the cut line for a particular load case.

        Parameters
        ----------
        plateList : list
            List of plates IDs. a) All plates in model, b) plates through which
        nplates : int
            No of plates in plateList
        loadIdList : list
            The load cases for plate analysis
        startNode : list
            List of x, y, z values of the start node at indexes 0, 1 and 2.
        endNode : list
            List of x, y, z values of the end node at indexes 0, 1 and 2.
        isTransformForceToGlobal : int
            1: return force in Global System 0: return forces in local system o
        firstNode : int
            Node no representing the origin point for building the surface axis
        secondNode : int
            Node no representing the second point for building the surface axis
        thirdNode : int
            Node no representing the third point for building the surface axis
        Returns
        -------
        list
            List of resultant forces consisting Fx, Fy, Fz, Mx, My, Mz (in same

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> plate_list = staad_obj.Geometry.GetPlateList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> resultant_force_along_line_for_plate_list = staad_obj.Output.GetResu
        """
        result = [make_safe_array_double(6)]
        ref_result = make_variant_vt_ref(result, automation.VT_ARRAY | automati
        plate_list = make_safe_array_long_input(plateList)
        safe_loadIdList = make_safe_array_long_input(loadIdList)
        safe_startNode = make_safe_array_double_input(startNode)
        safe_endNode = make_safe_array_double_input(endNode)
        vt_plate_list = make_variant_vt_ref(plate_list, automation.VT_ARRAY | au
        vt_loadIdList = make_variant_vt_ref(safe_loadIdList, automation.VT_ARRAY
        vt_startNode = make_variant_vt_ref(safe_startNode, automation.VT_ARRAY
        vt_endNode = make_variant_vt_ref(safe_endNode, automation.VT_ARRAY | au
        result = self._output.GetResultantForceAlongLineForPlateList(vt_plate_l
        if not result:
            raise_os_error_if_error_code(-1)
        return list(ref_result[0])
```

[docs]

```
    def GetResultantForceAlongLineForParametricSurface(self, parametricSurfaceNa
        """
        Gets forces and moments along the cut line for a particular load case.

        Parameters
        ----------
        parametricSurfaceName : str
            Name of the parametric surface.
        nplates : int
            Number of plates in plateList.
        loadId : int
            The load case for plate analysis.
        startNode : list
            List of x, y, z values of the start node at indexes 0, 1 and 2.
        endNode : list
            List of x, y, z values of the end node at indexes 0, 1 and 2.
        facingNode : list
            List of x, y, z values of the facing node at indexes 0, 1 and 2.
        isTransformForceToGlobal : int
            1: return force in Global System, 0: return forces in local system 
        firstNode : int
            Node no representing the origin point for building the surface axis
        secondNode : int
            Node no representing the second point for building the surface axis
        thirdNode : int
            Node no representing the third point for building the surface axis s

        Returns
        -------
        list
            List of resultant forces consisting Fx, Fy, Fz, Mx, My, Mz (in same

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> surface_name = "WALL 1"
        >>> plate_list = staad_obj.Geometry.GetPlateList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> result = staad_obj.Output.GetResultantForceAlongLineForParametricSur
        """
        result = make_safe_array_double(6)
        ref_result = make_variant_vt_ref(result, automation.VT_ARRAY | automatio
        safe_startNode = make_safe_array_long_input(startNode)
        safe_endNode = make_safe_array_long_input(endNode)
        safe_facingNode = make_safe_array_long_input(facingNode)
        vt_startNode = make_variant_vt_ref(safe_startNode, automation.VT_ARRAY
        vt_endNode = make_variant_vt_ref(safe_endNode, automation.VT_ARRAY | au
        vt_facingNode = make_variant_vt_ref(safe_facingNode, automation.VT_ARRAY
        result = self._output.GetResultantForceAlongLineForParametricSurface(pa
        if not result:
            raise_os_error_if_error_code(-1)
        return list(ref_result[0])
```

[docs]

```
    def GetPlateStressAtPoint(self, plateNo: int, loadNo: int, stressPoint: lis
        """
        Get stresses values at a point on a specified plate.

        Parameters
        ----------
        plateNo : int
            Number of the plate.
        loadNo : int
            Load number for which stress is requested
        stressPoint : list
            The coordinate at which the stress is required in global axes as an
        facingPoint : int
            x, y, z values of the facing node at indexes 0, 1 and 2. API always
            Definition of facingPoint: It is the node which sits on the tip of a

        Returns
        -------
        list
            List of stresses values at a point on a specified plate containing t
                +-------------+------------------------+
                | Array Index | stress Type            |
                +=============+========================+
                | 0           | None                   |
                +-------------+------------------------+
                | 1           | MaxAbs                 |
                +-------------+------------------------+
                | 2           | TopMax                 |
                +-------------+------------------------+
                | 3           | TopMin                 |
                +-------------+------------------------+
                | 4           | TopTauMax              |
                +-------------+------------------------+
                | 5           | BotMax                 |
                +-------------+------------------------+
                | 6           | BotMin                 |
                +-------------+------------------------+
                | 7           | BotTauMax              |
                +-------------+------------------------+
                | 8           | MaxVM                  |
                +-------------+------------------------+
                | 9           | VMTopMax               |
                +-------------+------------------------+
                | 10          | VMBotMax               |
                +-------------+------------------------+
                | 11          | MaxTresca              |
                +-------------+------------------------+
                | 12          | TopTresca              |
                +-------------+------------------------+
                | 13          | BotTresca              |
                +-------------+------------------------+
                | 14          | FX                     |
                +-------------+------------------------+
                | 15          | FY                     |
                +-------------+------------------------+
                | 16          | FXY                    |
                +-------------+------------------------+
                | 17          | MX                     |
                +-------------+------------------------+
                | 18          | MY                     |
                +-------------+------------------------+
                | 19          | MZ                     |
                +-------------+------------------------+
                | 20          | QX                     |
                +-------------+------------------------+
                | 21          | QY                     |
                +-------------+------------------------+
                | 22          | Global                 |
                +-------------+------------------------+
                | 23          | GlobalMembraneStresses |
                +-------------+-----------------------+
                | 24          | GlobalshearStresses   |
                +-------------+-----------------------+
                | 25          | BasePres              |
                +-------------+-----------------------+
                | 26          | CombXTop              |
                +-------------+-----------------------+
                | 27          | CombYTop              |
                +-------------+-----------------------+
                | 28          | CombXYTop             |
                +-------------+-----------------------+
                | 29          | CombXBot              |
                +-------------+-----------------------+
                | 30          | CombYBot              |
                +-------------+-----------------------+
                | 31          | CombXYBot             |
                +-------------+-----------------------+

            Example
            -------
            >>> from openstaadpy import os_analytical
            >>>
            >>> staad_obj = os_analytical.connect()
            >>>
            >>> surface_name = "WALL 1"
            >>> plate_list = staad_obj.Geometry.GetPlateList()
            >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
            >>> staad_obj.AnalyzeEx(1, 0, 1)
            >>>
            >>> plate_stress_at_point = staad_obj.Output.GetPlateStressAtPoint(plate
            """
            safe_stress_point = make_safe_array_double_input(stressPoint)
            safe_facing_point = make_safe_array_double_input(facingPoint)
            safe_stresses = make_safe_array_double(32)
            ref_stress_point = make_variant_vt_ref(safe_stress_point, automation.VT_
            ref_facing_point = make_variant_vt_ref(safe_facing_point, automation.VT_
            ref_stresses = make_variant_vt_ref(safe_stresses, automation.VT_ARRAY |
            result = self._output.GetPlateStressAtPoint(plateNo, loadNo, ref_stress_
            if not result:
                raise_os_error_if_error_code(-1)
            return list(ref_stresses[0])
```

[docs]

```
    def GetTimeHistoryIntegrationStepInfo(self):
        """
        Gets the time-step (secs) used for time-history integration.

        Returns
        -------
        float
            Time step used for the integration in seconds.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>>
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> delta, nsteps = staad_obj.Output.GetTimeHistoryIntegrationStepInfo()
        """
        time_step = make_safe_array_double(1)
        ref_time_step = make_variant_vt_ref(time_step, automation.VT_R8)
        nsteps = self._output.GetTimeHistoryIntegrationStepInfo(ref_time_step)
        if nsteps < 0:
            raise_os_error_if_error_code(-1)
        return float(ref_time_step[0]), nsteps
```

[docs]

```
    def GetTimeHistoryResponseAtTime(self, load_case: int, node_no: int, dof_no
        """
        Gets the response at a specific time within the integration time span at

        Parameters
        ----------
        load_case : int
            Load case number for future use. Use 0 at present.
        node_no : int
            Node number where the response is sought.
        dof_no : int
            Degrees of freedom define as DegreesOfFreedom enum:
                +--------+--------------------+
                | Value  | Degrees Of Freedom |
                +========+====================+
                | Fx = 1 | DegreesOfFreedom.Fx |
                +--------+--------------------+
                | Fy = 2 | DegreesOfFreedom.Fy |
                +--------+--------------------+
                | Fz = 3 | DegreesOfFreedom.Fz |
                +--------+--------------------+
                | Mx = 4 | DegreesOfFreedom.Mx |
                +--------+--------------------+
                | My = 5 | DegreesOfFreedom.My |
                +--------+--------------------+
                | Mz = 6 | DegreesOfFreedom.Mz |
                +--------+--------------------+
        response_type : int
            Response type, i.e., displacement, velocity, or acceleration defined
                +------------------+--------------------------------------+
                | Value            | Response Type                        |
                +==================+======================================+
                | displacement = 0 | TimeHistoryResponseType.dispResponse |
                +------------------+--------------------------------------+
                | velocity = 1     | TimeHistoryResponseType.velResponse  |
                +------------------+--------------------------------------+
                | acceleration = 2 | TimeHistoryResponseType.acclResponse |
                +------------------+--------------------------------------+
        at_time : int
            Time in seconds for which the response is sought.


        Returns
        -------
        list of float
            List returning the responses at the integration steps. The size of t

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> node_list = staad_obj.Geometry.GetNodeList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> time_history_response_at_time = staad_obj.Output.GetTimeHistoryRespo
        """
        response = make_safe_array_double(1)
        ref_response = make_variant_vt_ref(response, automation.VT_R8)
        result = self._output.GetTimeHistoryResponseAtTime(load_case, node_no, d
        if result < 0:
            raise_os_error_if_error_code(-1)
        return ref_response[0], result
```

[docs]

```
def GetTimeHistoryResponse(self, load_case: int, node_no: int, dof_no: int,
    """
    Gets the time-history responses of DOF at specified node in the VARIANT

    Parameters
    ----------
    load_case : int
        Load case number for future use. Use 0 at present.
    node_no : int
        Node number where the response is sought.
    dof_no : int
        Degrees of freedom define as DegreesOfFreedom enum:
            +--------+--------------------+
            | Value  | Degrees Of Freedom  |
            +========+====================+
            | Fx = 1 | DegreesOfFreedom.Fx |
            +--------+--------------------+
            | Fy = 2 | DegreesOfFreedom.Fy |
            +--------+--------------------+
            | Fz = 3 | DegreesOfFreedom.Fz |
            +--------+--------------------+
            | Mx = 4 | DegreesOfFreedom.Mx |
            +--------+--------------------+
            | My = 5 | DegreesOfFreedom.My |
            +--------+---------------------+
            | Mz = 6 | DegreesOfFreedom.Mz |
            +--------+---------------------+
    response_type : int
        Response type, i.e., displacement, velocity, or acceleration defined
            +------------------+--------------------------------------+
            | Value            | Response Type                        |
            +==================+======================================+
            | displacement = 0 | TimeHistoryResponseType.dispResponse |
            +------------------+--------------------------------------+
            | velocity = 1     | TimeHistoryResponseType.velResponse  |
            +------------------+--------------------------------------+
            | acceleration = 2 | TimeHistoryResponseType.acclResponse |
            +------------------+--------------------------------------+

    Returns
    -------
    float
        Response value.

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>> node_list = staad_obj.Geometry.GetNodeList()
    >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> time_history_response = staad_obj.Output.GetTimeHistoryResponse(loac
    """
    delta, nsteps = self.GetTimeHistoryIntegrationStepInfo()
    response = make_safe_array_double(nsteps)
    ref_response = make_variant_vt_ref(response, automation.VT_ARRAY | autor
    result = self._output.GetTimeHistoryResponse(load_case, node_no, dof_no
    if result != 1:
        raise_os_error_if_error_code(-1)
    return ref_response[0]
```

[docs]

```
    def GetTimeHistoryResponseMinMax(self, load_case: int, node_no: int, dof_no
        """
        Gets the min/max time-history responses of DOF at specified node.

        Parameters
        ----------
        load_case : int
            Use 0 at present.
        node_no : int
            Node number where the response is sought.
        dof_no : int
            Degrees of freedom define as DegreesOfFreedom enum:
```

```
            +--------+--------------------+
            | Value  | Degrees Of Freedom |
            +========+====================+
            | Fx = 1 | DegreesOfFreedom.Fx |
            +--------+--------------------+
            | Fy = 2 | DegreesOfFreedom.Fy |
            +--------+--------------------+
            | Fz = 3 | DegreesOfFreedom.Fz |
            +--------+--------------------+
    response_type : int
        Response type, i.e., displacement, velocity, or acceleration definec
            +------------------+--------------------------------------+
            | Value            | Response Type                        |
            +==================+======================================+
            | displacement = 0 | TimeHistoryResponseType.dispResponse |
            +------------------+--------------------------------------+
            | velocity = 1     | TimeHistoryResponseType.velResponse  |
            +------------------+--------------------------------------+
            | acceleration = 2 | TimeHistoryResponseType.acclResponse |
            +------------------+--------------------------------------+
    Returns
    -------
    tuple
        Tuple constisiting of maximum response, the time when the maximum re

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>> node_list = staad_obj.Geometry.GetNodeList()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> time_min_max = staad_obj.Output.GetTimeHistoryResponseMinMax(0, node
    """
    response_max_val = make_safe_array_double(1)
    response_min_val = make_safe_array_double(1)
    time_max_val = make_safe_array_double(1)
    time_min_val = make_safe_array_double(1)
    ref_responseMax = make_variant_vt_ref(response_max_val, automation.VT_R8
    ref_responseMin = make_variant_vt_ref(response_min_val, automation.VT_R8
    ref_timeMax = make_variant_vt_ref(time_max_val, automation.VT_R8)
    ref_timeMin = make_variant_vt_ref(time_min_val, automation.VT_R8)
    result =self._output.GetTimeHistoryResponseMinMax(load_case, node_no, do
    if (result < 1):
        raise_os_error_if_error_code(-1)
    return ref_responseMax[0], ref_timeMax[0], ref_responseMin[0], ref_timeM
```

[docs]

```
def GetMemberSteelDesignResults(self, beamNo: int):
    """
    Gets steel design results for the specified member. This method will ret
```

```
        Parameters
        ----------
        beamNo : int
            Id of the member for which design results should be retrieved.

        Returns
        -------
        tuple
            Tuple of steel design results containing the design code name, the d

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> beam_list = staad_obj.Geometry.GetBeamList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> design_code_name, design_status, critical_ratio, allowable_ratio, cr
        """
        designcode = create_bstr()
        designstatus = create_bstr()
        criticalratio = make_safe_array_double(1)
        allowableratio = make_safe_array_double(1)
        criticalloadcase = make_safe_array_long(1)
        criticalsection = make_safe_array_double(1)
        criticalclause = create_bstr()
        designsection = create_bstr()
        designforce = make_safe_array_double(3)
        klbyr = make_safe_array_double(1)
        ref_designcode = make_byref(designcode)
        ref_designstatus = make_byref(designstatus)
        ref_criticalratio = make_variant_vt_ref(criticalratio, automation.VT_R8)
        ref_allowableratio = make_variant_vt_ref(allowableratio, automation.VT_I
        ref_criticalloadcase = make_variant_vt_ref(criticalloadcase, automation
        ref_criticalsection = make_variant_vt_ref(criticalsection, automation.VT
        ref_criticalclause = make_byref(criticalclause)
        ref_designsection = make_byref(designsection)
        ref_designforce = make_variant_vt_ref(designforce , automation.VT_ARRAY
        ref_klbyr = make_variant_vt_ref(klbyr , automation.VT_R8)
        result = self._output.GetMemberSteelDesignResults(beamNo, ref_designcode
        if (result < 0):
            raise_os_error_if_error_code(-1)
        return designcode.value, designstatus.value, ref_criticalratio[0], ref_a
```

[docs]

```
    def GetMemberSteelDesignMinFailureRatio(self):
        """
        Gets the minimum failure ratio across all beams in the model.

        Returns
        -------
        float
```

```
            The minimum failure ratio.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> member_steel_design_min_failure_ratio = staad_obj.Output.GetMemberSt
        """
        ratio = make_safe_array_double(1)
        ref_ratio = make_variant_vt_ref(ratio, automation.VT_R8)
        result = self._output.GetMemberSteelDesignMinFailureRatio(ref_ratio)
        if not result:
            raise_os_error_if_error_code(-1)
        return ref_ratio[0]
```

[docs]

```
    def GetMemberSteelDesignMaxFailureRatio(self):
        """
        Gets the maximum failure ratio across all beams in the model.

        Returns
        -------
        float
            The maximum failure ratio.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> member_steel_design_max_failure_ratio = staad_obj.Output.GetMemberSt
        """
        ratio = make_safe_array_double(1)
        ref_ratio = make_variant_vt_ref(ratio, automation.VT_R8)
        result = self._output.GetMemberSteelDesignMaxFailureRatio(ref_ratio)
        if not result:
            raise_os_error_if_error_code(-1)
        return ref_ratio[0]
```

[docs]

```
    def IsMultipleMemberSteelDesignResultsAvailable(self):
        """
        Checks whether steel design results from multiple design block can be ex
        If true, then relevant multiple steel design parameters like GetMultiple
        Currently, this facility is limited to AISC 360-16 code only. For furthe
        Returns
        -------
        bool
            returns true (for boolean variable, for long variable, return value

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> is_multiple_member_steel_design_results_available = staad_obj.Output
        """
        return bool(self._output.IsMultipleMemberSteelDesignResultsAvailable())
```

[docs]

```
    def GetSteelDesignParameterBlockCount(self):
        """
        Gets the count of steel design parameter blocks in the model. This funct

        Returns
        -------
        int
            Returns the count of steel design parameter blocks.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> steel_design_parameter_block_count = staad_obj.Output.GetSteelDesign
        """
        return self._output.GetSteelDesignParameterBlockCount()
```

[docs]

```
    def GetSteelDesignParameterBlockNameByIndex(self, index: int):
        """
        Gets steel design parameter name at the specified index. This function i

        Parameters
        ----------
        index : int
            The index value of steel design parameter block list. Note, the inde

        Returns
        -------
        str
            Steel design parameter block name.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> countOfParamBlocks = staad_obj.Output.GetSteelDesignParameterBlockCo
        >>> if (countOfParamBlocks > 0) :
        >>>     param_blk_name = staad_obj.Output.GetSteelDesignParameterBlockNa
        """
        name = create_bstr()
        ref_name = make_byref(name)
        result = self._output.GetSteelDesignParameterBlockNameByIndex(index, re
        if not result:
            raise_os_error_if_error_code(-1)
        return name.value
```

[docs]

```
    def GetMultipleMemberSteelDesignRatio(self, param_blk_name: str, beam_no: in
        """
        Gets the critical steel design ratio for a steel member. This function i

        Parameters
        ----------
        param_blk_name : str
            Steel design parameter block name.
        beam_no : int
            Beam number ID.

        Returns
        -------
        float
            Returns the critical steel design ratio.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> countOfParamBlocks = staad_obj.Output.GetSteelDesignParameterBlockCo
        >>> if (countOfParamBlocks > 0) :
        >>>     param_blk_name = staad_obj.Output.GetSteelDesignParameterBlockNa
        >>>     beam_list = staad_obj.Geometry.GetBeamList()
        >>>     staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>>     multiple_member_steel_design_ratio = staad_obj.Output.GetMultipl
        """
        ratio = make_safe_array_double(1)
        ref_ratio = make_variant_vt_ref(ratio, automation.VT_R8)
        result = self._output.GetMultipleMemberSteelDesignRatio(param_blk_name,
        if not result:
            raise_os_error_if_error_code(-1)
        return ref_ratio[0]
```

[docs]

```
    def GetMultipleMemberSteelDesignResults(self, param_blk_name: str, beam_no:
        """
        Gets the critical steel design result information for a steel member.

        Parameters
        ----------
        param_blk_name : str
            Steel design parameter block name.
        beam_no : int
            Beam number ID.

        Returns
        -------
        tuple
            Tuple consisting of the design code name, the design status (pass or

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> countOfParamBlocks = staad_obj.Output.GetSteelDesignParameterBlockCo
        >>> if (countOfParamBlocks > 0) :
        >>>     param_blk_name = staad_obj.Output.GetSteelDesignParameterBlockNa
        >>>     beam_list = staad_obj.Geometry.GetBeamList()
        >>>     staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>>     multiple_member_steel_design_results = staad_obj.Output.GetMulti
        """
        designcode = create_bstr()
        designstatus = create_bstr()
        criticalratio = make_safe_array_double(1)
        allowableratio = make_safe_array_double(1)
        criticalloadcase = make_safe_array_long(1)
        criticalclause = create_bstr()
        designsection = create_bstr()
        ref_designcode = make_byref(designcode)
        ref_designstatus = make_byref(designstatus)
        ref_criticalratio = make_variant_vt_ref(criticalratio, automation.VT_R8
        ref_allowableratio = make_variant_vt_ref(allowableratio, automation.VT_
        ref_criticalloadcase = make_variant_vt_ref(criticalloadcase, automation
        ref_criticalclause = make_byref(criticalclause)
        ref_designsection = make_byref(designsection)
        result = self._output.GetMultipleMemberSteelDesignResults(param_blk_name
        if result != 1:
            raise_os_error_if_error_code(-1)
        return designcode.value, designstatus.value, ref_criticalratio[0], ref_a
```

[docs]

```
    def GetMultipleMemberSteelDesignMaxRatio(self, beamNo: int):
        """
        Gets the maximum critical steel design ratio across all parameter blocks

        Parameters
        ----------
        beamNo : int
            Beam number ID.

        Returns
        -------
        float
            Returns the maximum critical steel design ratio.

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> beam_list = staad_obj.Geometry.GetBeamList()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> multiple_member_steel_design_max_ratio = staad_obj.Output.GetMultipl
        """
        ratio = make_safe_array_double(1)
        ref_ratio = make_variant_vt_ref(ratio, automation.VT_R8)
        result = self._output.GetMultipleMemberSteelDesignMaxRatio(beamNo, ref_
        if result != 1:
            raise_os_error_if_error_code(-1)
        return ref_ratio[0]
```

[docs]

```
    def GetAllPlateCenterPrincipalStressesAndAnglesEx(self, plateNo: int, loadC
        """
        Get all plate center principal stresses and angles (extended).

        Parameters
        ----------
        plateNo : int
            Plate number ID.
        loadCaseNo : int
            Load case number ID.

        Returns
        -------
        tuple
            Tuple of principal stresses values list and angle value list. They a

            Principal Stresses list :
                +---------------+-------------------------------------------+
                | Variable      | Description                              |
                +===============+==========================================+
                | pdStresses[0] | Top-Maximum in-plane Principal stress    |
                +---------------+------------------------------------------+
                | pdStresses[1] | Top-Minimum in-plane Principal stress    |
                +---------------+------------------------------------------+
                | pdStresses[2] | Top-Maximum in-plane Shear stress        |
                +---------------+------------------------------------------+
                | pdStresses[3] | Bottom-Maximum in-plane Principal stress |
                +---------------+------------------------------------------+
                | pdStresses[4] | Bottom-Minimum in-plane Principal stress |
                +---------------+------------------------------------------+
                | pdStresses[5] | Bottom-Maximum in-plane Shear stress     |
                +---------------+------------------------------------------+

            Angles List :
                +-------------+------------------------------------------------
                | Variable    | Description
                +=============+================================================
                | pdAngles[0] | Top-Angle which determines direction of maximum
                +-------------+------------------------------------------------
                | pdAngles[1] | Bottom-Angle which determines direction of maxi
                +-------------+------------------------------------------------

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> plate_list = staad_obj.Geometry.GetPlateList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> principal_stress_list, angle_list = staad_obj.Output.GetAllPlateCent
        """
        re_principal_stress = make_safe_array_double(6)
        vt_principal_stress = make_variant_vt_ref(re_principal_stress, automati
        re_angle_list = make_safe_array_double(2)
        vt_angle_list = make_variant_vt_ref(re_angle_list, automation.VT_ARRAY
        result = self._output.GetAllPlateCenterPrincipalStressesAndAnglesEx(pla
        if not result:
            raise_os_error_if_error_code(-1)
        return list(vt_principal_stress[0]), list(vt_angle_list[0])
```

[docs]

```
def GetPMemberEndForces(self, memberNo: int, end: int, loadCaseNo: int, Loc
    """
    Gets member end forces for specified physical member number, member end

    Parameters
    ----------
    memberNo : int
        Member number ID.
    end : int
        Member End (0 for starting and 1 for ending).
    loadCaseNo : int
        Load Case reference ID.
    LocalOrGlobal : int
        Results returned in either local or global axes. 0= Local, 1= Global

    Returns
    -------
    list
        List of force of Member End in LOCAL coordinates in terms of FX, FY,

    Example
    -------
    >>> from openstaadpy import os_analytical
    >>>
    >>> staad_obj = os_analytical.connect()
    >>> plate_list = staad_obj.Geometry.GetPlateList()
    >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
    >>> staad_obj.AnalyzeEx(1, 0, 1)
    >>>
    >>> p_member_end_forces = staad_obj.Output.GetPMemberEndForces(plate_lis
    """
    vt_Forces = make_safe_array_double(6)
    re_Forces = make_variant_vt_ref(vt_Forces, automation.VT_ARRAY | automat
    result = self._output.GetPMemberEndForces(memberNo, end, loadCaseNo, re_
    if not result:
        raise_os_error_if_error_code(-1)
    return list(re_Forces[0])
```

[docs]

```
    def GetPMemberIntermediateForcesAtDistance(self, memberNo: int, distance: in
        """
        Gets sectional forces and moments for specified physical member number,

        Parameters
        ----------
        memberNo : int
            Physical Member number ID.
        distance : int
            Distance from the starting end of the member.
        loadCaseNo : int
            Load Case reference ID.
        Returns
        -------
        list
            List of 6 elements consisting of Section axial force, Shear force in

        Example
        -------
        >>> from openstaadpy import os_analytical
        >>>
        >>> staad_obj = os_analytical.connect()
        >>> plate_list = staad_obj.Geometry.GetPlateList()
        >>> load_cases = staad_obj.Load.GetPrimaryLoadCaseNumbers()
        >>> staad_obj.AnalyzeEx(1, 0, 1)
        >>>
        >>> p_member_end_forces = staad_obj.Output.GetPMemberIntermediateForcesA
        """
        vt_Forces = make_safe_array_double(6)
        re_Forces = make_variant_vt_ref(vt_Forces, automation.VT_ARRAY | automat
        result = self._output.GetPMemberIntermediateForcesAtDistance(memberNo, 
        if not result:
            raise_os_error_if_error_code(-1)
        return list(re_Forces[0])
```