Load

Contents

class `openstaadpy.os_analytical.osload.OSLoad` [source]

Bases: `object`

`AddAutoCombinationRepeat`(*varCode: str, varCategory: str, varLoadList: list, varStartLoadCaseNo: int, varGeneratedLCS: int, bVarReference: bool, bVarNotional: bool, dVarNotionalLoadFactor: float, bVarGB50017: bool, nVarFloor: int, bVarX: bool, bVarNegtiveX: bool, bVarZ: bool, bVarNegtiveZ: bool*) [source]

Automatically adds repeat load based on assigned design code and Category.

Parameters:

- **varCode** (*bool*) – Load Combination Code string name (refer to “Codes.ini”)
- **varCategory** (*bool*) – Load Combination Category string name (refer to corresponding rule ini file defined in “Codes.ini”)
- **varLoadList** (*List of int*) – List of Load case reference IDs. If the array is either null or empty then all load cases in current model will be considered.
- **varStartLoadCaseNo** (*int*) – (Repeat Load) Load case reference ID with which automatically generation starts. If nStartLoadCaseNo is valid, auto repeat load will be created from the provided ID. If nStartLoadCaseNo is invalid Load Case ID/already present Load Case ID, the repeat load would automatically generated from next available Load Case ID and nStartLoadCaseNo will be returned/updated with this ID.
- **varGeneratedLCS** (*int*) – (Repeat Load) The counts of automatically generated repeat loads.
- **bVarReference** (*bool*) – Whether include Reference load
- **bVarNotional** (*bool*) – Whether include Notional load. If it’s True but all Directions are Flase, return -1.
- **dVarNotionalLoadFactor** (*float*) – If bVarNotional is valid, the value of Notional load factor
- **bVarGB50017** (*bool*) – Consider Notional load factor per GB 50017 Design code
- **nVarFloor** (*int*) – The count of floor, it is valid when bVarGB50017 is True only
- **bVarX** (*bool*) – Consider X Direction of Notional Load
- **bVarNegtiveX** (*bool*) – Consider -X Direction of Notional Load
- **bVarZ** (*bool*) – Consider Z Direction of Notional Load
- **bVarNegtiveZ** (*bool*) – Consider -Z Direction of Notional Load

Returns:

- *int* – Returns 0 if successful. Returns -1 if unsuccessful
- *Notes* –
  - The default path of Codes.ini under “%localappdata%BentleyEngineeringSTAAD.Pro 2023DefaultLanguageen”.

`AddAutoLoadCombinations`(*loadCombCode: str, loadCombCategory: str, loadList: list*) [source]

Automatically adds load combination based on assigned design code and Category.

Parameters:

- **loadCombCode** (*str*) – Load Combination Code string name (refer to “Codes.ini”)
- **loadCombCategory** (*str*) – Load Combination Category string name (refer to corresponding rule ini file defined in “Codes.ini”)

- **loadList** (*list of int*) – Load case reference ID(s), Array of Load case numbers. If the array is either null or empty then all load cases in current model will be considered

Returns:

- *int* – Returns load case reference ID with which automatically load combination generation starts. If nStartLoadCaseNo is valid, auto load combinations will be created from the provided ID. IfnStartLoadCaseNo is invalid Load Case ID already present Load Case ID, load combinations would automatically generated from next available Load Case ID and nStartLoadCaseNo will be returned/updatedwith this ID.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddAutoLoadCombinations("AISC 9th Ed","2.3 LRFD General", [1, 2, 3])*

`AddDirectAnalysisDefinitionParameter(pParamType: int, members: list, param: float)` [source]

Adds Direct Analysis Definition (FLEX,AXIAL parameters).

Parameters:

- **pParamType** (*int*) –

  Integer indicating type of direct analysis parameter to be added. Integer value should be taken from following table :

Value	AnalysisCommand
FLEX = 0	DirectAnalysisParameterTypes.FLEX
AXIAL = 2	DirectAnalysisParameterTypes.AXIAL

- **members** (*list of int*) – List of Member IDs
- **param** (*float*) – FLEX parameter value. [For AXIAL this value is Not Applicable. Should pass 0(zero)]

Returns:

- *bool* – Returns TRUE if successful Returns FALSE if unsuccessful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddDirectAnalysisDefinitionParameter(0, [1, 2], 0.5)*
- *True*

`AddElementHydrostaticPressure(plateIds: list, varLoadDirection: int, varInterpolateDirection: int, varMinLoad: float, varMaxLoad: float)` [source]

Adds Hydrostatic pressure loading to plate elements.

Parameters:

- **plateIds** (*list of int*) – List of plate IDs.
- **varLoadDirection** (*int*) – Load direction: (= 3 to 6 for LocalZ, GlobalX, GlobalY, GlobalZ, respectively)
- **varInterpolateDirection** (*int*) – Interpolate along Global Axis(Int or Long), valid direction codes are 1, 2, 3 for Interpolate along Global X, Y, Z. No other direction is a valid input.
- **varMinLoad** (*float*) – Minimum Pressure load
- **varMaxLoad** (*float*) – Maximum Pressure load

Returns:

True if successfully. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddElementHydrostaticPressure([1], 3, 1, 0.0, 10.0)
```

AddElementPressure(*plateIds: list, varDirection: int, varPressure: float, varX1: float, varY1: float, varX2: float, varY2: float*) [source]

Adds pressure load to plate elements.

Parameters:

- **plateIds** (*list of int*) – List of plate IDs.
- **varDirection** (*int*) – Load direction: (1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY and ProjectedZ respectively).
- **varPressure** (*float*) – Magnitude of the pressure or concentrate load on the element.
- **varX1** (*float*) – Top-Left coordinate X (local).
- **varY1** (*float*) – Top-Left coordinate Y (local).
- **varX2** (*float*) – Bottom-Right coordinate X (local).
- **varY2** (*float*) – Bottom-Right coordinate Y (local).
- **Notes**
- **X1** (*- If*)
- **Y1** (*Pressure applied over the area between (X1 , Y1) and (X2 , Y2) measured from the center of plate(s) in the local axis system.*)
- **0** (*but X2 and Y2 are*)
- **element.** (*the pressure is applied over the full area of the*)
- **X1**
- **Y1**
- **0**
- **0**
- **0**

Returns:

Returns 0 if OK. Returns -1 if General error. Returns -8001 if Load direction is invalid.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddElementPressure([1], 3, 5.0, 0.0, 0.0, 1.0, 1.0)
```

AddElementTrapPressureEx(*PlateIDs: list, LoadDirection: int, LoadVaryDirection: int, StartPressure: float, EndPressure: float, Pressure3: float, Pressure4: float*) [source]

Adds trapezoidal pressure loading to plate elements.

Parameters:

- **PlateNo** (*list of int*) – List of Plate IDs.
- **LoadDirection** (*int*) – Load direction: (= 3 to 6 for LocalZ, GlobalX, GlobalY, GlobalZ, respectively)
- **LoadVaryDirection** (*int*) – Load varying direction: (= 1, 2 ,3 for X, Y and JOINT respectively)
- **StartPressure** (*float*) – Pressure at loading starting point.(Node1 when JOINT is selected)
- **EndPressure** (*float*) – Pressure at loading ending point.(Node2 when JOINT is selected)

- **Pressure3** (*float*) – Pressure at loading point.(applicable only when JOINT is selected)
- **Pressure4** (*float*) – Pressure at loading point.(applicable only when JOINT is selected)

Returns:

True if OK. False if any General error.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddElementTrapPressureEx([1], 3, 1, 100.0, 200.0, 150.0, 250.0)
```

AddLoadAndFactorToCombination(*loadCombNo: int, loadNo: int, factor: float*) [source]

Adds a primary load case with specified multiplication factor to an existing load combination.

Parameters:

- **loadCombNo** (*int*) – Load Combination Number.
- **loadNo** (*int*) – Load Case Reference ID.
- **factor** (*float*) – Multiplication factor for the specified primary load case.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddLoadAndFactorToCombination(1, 2, 1.0)
```

AddLoadCasesToEnvelop(*varEnvNo: int, varLoadCaseList: list*) [source]

Adds a list of primary load case(s) to an existed load envelop.

Parameters:

- **varEnvNo** (*int*) – Load Envelop reference ID
- **varLoadCaseList** (*list of int*) – Load cases reference IDs list.

Returns:

- *int* – Returns 1 if OK. Returns 0 if general error.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddLoadCasesToEnvelop(1, [1, 2, 3])*

AddMemberAreaLoad(*beamIds: list, load: float*) [source]

Adds AREA LOAD to beams.

Parameters:

- **beamIds** (*list of int*) – List of Beam IDs.
- **load** (*float*) – Magnitude of the load value.

Returns:

Returns 0 if OK. Returns -1 if General error.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddMemberAreaLoad([1], 5.0)
```

AddMemberConcForce(*beamIds: list, varDirection: int, varForce: float, varD1: float, varD2: float*)

Adds a concentrated force to the specified beams. [source]

Parameters:

- **beamIds** (*list of int*) – List of beam IDs.
- **varDirection** (*int*) – Load direction (1 to 6 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ respectively).
- **varForce** (*float*) – Magnitude of the concentrate force in current units.
- **varD1** (*float*) – Distance from the start of the member to concentrated force.
- **varD2** (*float*) – Perpendicular distance from the member shear center to the local plane of loading.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberConcForce([1], 1, 20.0, 2.5, 0)
>>> print(result)
```

AddMemberConcMoment(*beamIds: list, varDirection: int, varMoment: float, varD1: float, varD2: float*)

Adds a concentrated moment to the specified beams. [source]

Parameters:

- **beamIds** (*list of int*) – List of beam IDs.
- **varDirection** (*int*) – Load direction (1 to 6 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ respectively).
- **varMoment** (*float*) – Magnitude of the concentrate moment in current units.
- **varD1** (*float*) – Distance from the start of the member to concentrated moment.
- **varD2** (*float*) – Perpendicular distance from the member shear center to the local plane of loading.

Returns:

Returns True if successful.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberConcMoment([1], 2, 15.0, 3.0, 0)
>>> print(result)
```

AddMemberFixedEnd(*beamIds: list, loadStart: float, loadEnd: float*) [source]

Adds FIXED END LOAD to beams.

Parameters:

- **beamIds** (*list of int*) – List of Beam IDs.
- **loadStart** (*list of float*) – Load at starting point in form of array containing 6 elements corresponding to FX, FY, FZ, MX, MY & MZ respective to each index.
- **loadEnd** (*list of float*) – Load at end point in form of array containing 6 elements corresponding to FX, FY, FZ, MX, MY & MZ respective to each index.

Returns:

Returns True if successful.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddMemberFixedEnd([1], [1.0, 1.0, 1.0, 0, 0, 0], [1.0, 1.0, 1.0, 0, 0, 0])
```

AddMemberFloorLoad(*varPressure: float, varYMIN: float, varYMAX: float, varZMIN: float, varZMAX: float, varXMIN: float, varXMAX: float*) [source]

Automatically finds enclosed panels in the given boundary (specified using max and min X, Y, Z range inputs) and adds a FLOOR LOAD. Generated floor load is applied only in the Global X direction with YRANGE option.

Parameters:

- **varPressure** (*float*) – Magnitude of the pressure or concentrate load on the element.
- **varYMIN** (*float*) – Y range from which the load start (in global coordinate).
- **varYMAX** (*float*) – Y range at which the load end (in global coordinate).
- **varZMIN** (*float*) – Z range from which the load start (in global coordinate).
- **varZMAX** (*float*) – Z range at which the load end (in global coordinate).
- **varXMIN** (*float*) – X range from which the load start (in global coordinate).
- **varXMAX** (*float*) – X range at which the load end (in global coordinate).

Returns:

Returns 1 OK. Returns 0 General error. Returns -8001 Load direction is invalid.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddMemberFloorLoad(5.0, 0.0, 10.0, 0.0, 10.0, 0.0, 10.0)
```

AddMemberFloorLoadEx(*rangeType: int, loadDirection: int, pressure: float, grpOrOneWay: int, yMIN: float, yMAX: float, zMIN: float, zMAX: float, xMIN: float, xMAX: float*) [source]

Automatically finds enclosed panels in the given boundary (specified using max and min of X/Y /Z range inputs and varRange) and if varRange is 3 adds member group FLOOR LOAD (specified by GrpOrOneWay input). Otherwise adds a FLOOR LOAD with pressure (dPressure) in the Global X/Y/Z direction (as specified by Direction input) with RANGE option.

Parameters:

- **rangeType** (*int*) –

  Type of the Range :

Value	Range Type
0	X-RANGE
1	Y-RANGE
2	Z-RANGE
3	Group Load

- **loadDirection** (*int*) –

  Load direction :

Value	Direction
0	Global X
1	Global Y
2	Global Z

- **pressure** (*float*) – Magnitude of the pressure or concentrate load on the elemen
- **grpOrOneWay** (*int*) – One-Way Load (if it is either “” or “0”) or corresponding group name to add Floor Group Load (if it contains Group string name). Notes: - Group name should be of FLOOR group type.
- **yMIN** (*float*) – Y range from which the load start(in global coordinate).
- **yMAX** (*float*) – Y range at which the load end(in global coordinate).
- **zMIN** (*float*) – Z range from which the load start(in global coordinate).
- **zMAX** (*float*) – Z range at which the load end(in global coordinate).
- **xMIN** (*float*) – X range from which the load start(in global coordinate).
- **xMAX** (*float*) – X range at which the load end(in global coordinate).

Returns:

Returns 1 OK. Returns 0 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddMemberFloorLoadEx(0, 0, 5.0, 0, 0.0, 10.0, 0.0, 10.0, 0.0, 10.0)
```

AddMemberLinearVari(*memberIds: list[int], varDirection: int, varW1: float, varW2: float, varW3: float*) [source]

Adds LINEARLY VARYING load to beams.

Parameters:

- **memberIds** (*list of int*) – List of member IDs.
- **varDirection** (*int*) – Load direction (1 to 3 for LocalX, LocalY, LocalZ respectively).
- **varW1** (*float*) – Load at the start of the member.
- **varW2** (*float*) – Load at the end of the member.

- **varW3** (*float*) – Load in the middle of the member (for triangular load).

Returns:

Returns 0 OK. Returns -1 General error. Returns -8001 Load direction is invalid.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberLinearVari([1], 2, 2.0, 0.0, 0.0)
>>> print(result)
```

AddMemberTrapezoidal(*memberIds: list, varDirection: int, varW1: float, varW2: float, varD1: float, varD2: float*) [source]

Adds trapezoidal linearly varying load to beams.

Parameters:

- **memberIds** (*list of int*) – List of member IDs.
- **varDirection** (*int*) – Load direction (1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ respectively).
- **varW1** (*float*) – Load at the start of the member.
- **varW2** (*float*) – Load at the end of the member.
- **varD1** (*float*) – Distance from the start of the member to loading starting point.
- **varD2** (*float*) –

  Distance from the start of the member to loading stopping point.

  Notes: - If varD1 and varD2 are not given, the load is assumed to cover the full member length.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberTrapezoidal([1], 1, 5.0, 10.0, 0, 5)
>>> print(result)
```

AddMemberUniformForce(*beamIds: list, varDirection: int, varForce: float, varD1: float, varD2: float, varD3: float*) [source]

Adds a uniform force to the specified beams.

Parameters:

- **beamIds** (*list of int*) – List of beam IDs.
- **varDirection** (*int*) – Load direction (1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ respectively).
- **varForce** (*float*) – Magnitude of the uniform force.
- **varD1** (*float*) – Distance from the start of the member to the start of the load.
- **varD2** (*float*) – Distance from the start of the member to the end of the load.
- **varD3** (*float*) – Perpendicular distance from the member shear center to the local plane of loading.

Returns:

True if the uniform force was added successfully. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberUniformForce([1,2], 1, 5.0, 0, 5, 0)
>>> print(result)
```

AddMemberUniformMoment(*beamIds: list, varDirection: int, varMoment: float, varD1: float, varD2: float, varD3: float*) [source]

Adds a uniform moment to the specified beams.

Parameters:

- **beamIds** (*list of int*) – List of beam IDs.
- **varDirection** (*int*) – Load direction (1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ).
- **varMoment** (*float*) – Magnitude of the uniform moment.
- **varD1** (*float*) – Distance from the start of the member to the start of the load.
- **varD2** (*float*) – Distance from the start of the member to the end of the load.
- **varD3** (*float*) – Perpendicular distance from the member shear center to the local plane of loading.

Returns:

True if the uniform moment was added successfully.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddMemberUniformMoment([1], 2, 10.0, 0, 5, 0)
>>> print(result)
```

AddNodalLoad(*nodeIds: list, forceInXDir: float, forceInYDir: float, forceInZDir: float, momentInXDir: float, momentInYDir: float, momentInZDir: float*) [source]

Adds joint load to the specified node numbers.

Parameters:

- **nodeIds** (*list of int*) – List of node IDs to apply the joint load.
- **forceInXDir** (*float*) – Force in the X direction.
- **forceInYDir** (*float*) – Force in the Y direction.
- **forceInZDir** (*float*) – Force in the Z direction.
- **momentInXDir** (*float*) – Moment in the X direction.
- **momentInYDir** (*float*) – Moment in the Y direction.
- **momentInZDir** (*float*) – Moment in the Z direction.

Returns:

True if the joint load was added successfully. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddNodalLoad([1,2], 10, 0, 0, 0, 0, 0)
>>> print(result)
```

AddNotionalLoad(*varPrimaryLoadCaseList: list[int], varPLFactorList: list[float], varPLDirectionList: list[int], varReferenceLoadCaseList: list[int], varRLFactorList: list[float], varRLDirectionList: list[int]*) [source]

Creates a Notional load case using combinations of previously defined primary load cases and Reference load cases.

Parameters:

- **varPrimaryLoadCaseList** (*list of int*) – List of Primary load case reference number IDs
- **varPLFactorList** (*list of int*) – List of Multiplication factor of Primary load cases
- **varPLDirectionList** (*list of int*) –

  List of Directions of Primary load cases. Directions can be passed as following:

Direction	Integer
X OR GlobalLoadDirection X	1 4
Y OR GlobalLoadDirection Y	2 5
Z OR GlobalLoadDirection Z	3 6

- **varReferenceLoadCaseList** (*list of int*) – List of Reference load case reference number IDs
- **varRLFactorList** (*list of int*) – List of Multiplication factor of Reference load cases
- **varRLDirectionList** (*list of int*) –

  List of Directions of Reference load cases. Directions can be passed as following:

Direction	Integer
X OR GlobalLoadDirection X	1 4
Y OR GlobalLoadDirection Y	2 5
Z OR GlobalLoadDirection Z	3 6

Returns:

- *bool* – return True if successful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddNotionalLoad([1, 2], [1.0, 1.2], [1, 2], [3], [0.8], [3])*

AddRSLoad(*varType: int, varFactArray: list, varAccOrDis: int, varScale: float, varDampType: int, varDampFact: float, varLinOrLog: int, varMis: int, varMisFact: float, varZpa: int, varZpaFact: float, varFf1: int, varFf1Fact: float, varFf2: int, varFf2Fact: float, varSaveFlag: int, varPairs: int, varDispOrAccSet: list, varVals: list*) [source]

Add Response Spectrum Load

Parameters:

- **varType** (*int*) – Response Spectrum Load type(1=Srss, 2=Cqc, 3=Absolute, 4=Asce, 5=Ten, 6=Csm)
- **varFactArray** (*list of float*) – Factor List.
- **varAccOrDis** (*int*) – 1 for Acceleration or 0 for Displacement.
- **varScale** (*float*) – Scale
- **varDampType** (*int*) – Damp Type(1=DAMP, 2=CDAMP, 3 = MDAMP)
- **varDampFact** (*float*) – Damp Factor
- **varLinOrLog** (*int*) – Interpolation Type: 1 for Logarithmic or 0 for Linear.
- **varMis** (*int*) – Missing Mass(1 for checked, 0 for unchecked)
- **varMisFact** (*float*) – Missing Mass Factor
- **varZpa** (*int*) – ZPA(1 for checked, 0 for unchecked)
- **varZpaFact** (*float*) – ZPA Factor
- **varFf1** (*int*) – Ff1(1 for checked, 0 for unchecked)
- **varFf1Fact** (*float*) – Ff1 Factor
- **varFf2** (*int*) – Ff2(1 for checked, 0 for unchecked)
- **varFf2Fact** (*float*) – Ff2 Factor
- **varSaveFlag** (*int*) – Save Flag(1 for checked, 0 for unchecked)
- **varPairs** (*int*) – Disp Or Acc Set pair count
- **varDispOrAccSet** (*List of float*) – Disp or Acc list.
- **varVals** (*List of float*) – Value list.

Returns:

- *int* – Returns 1 if is successfully adds response spectrum load. Returns 0 if general error.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddRSLoad(1, [1.0, 2.0, 3.0], 1, 1.0, 1, 1.0, 1, 1, 1.0, 1, 1.0, 1, 1.0, 1, 1.0, 1, 1, [1.0, 2.0, 3.0], [1.0, 2.0, 3.0])*

AddReferenceLoad(*varRefLoadCaseNoIds: list[int], varFactorList: list[float]*) [source]

Adds a reference load item to the currently active load case.

Parameters:

- **varRefLoadCaseNoIds** (*list of int*) – List of reference load case number IDs from Reference Load Definitions.
- **varFactorList** (*list of float*) – List of corresponding load factors.

Returns:

- *int* – Reference load case number ID.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddReferenceLoad([1, 2, 3], [1.0, 2.0, 3.0])*

AddRepeatLoad(*varLoadCaseList: list, varFactorList: list*) [source]

Creates a primary load case using combinations of previously defined primary load cases.

Parameters:

- **varLoadCaseList** (*list of int*) – (Primary) load case reference number ID(s) array.
- **varFactorList** (*list of float*) – Multiplication factor array.

Returns:

- *int* – Returns 1 if Load Case is added successfully, 0 otherwise.
- *>>> from openstaadpy import os_analytical*

- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddRepeatLoad([1, 2, 3], [1.0, 2.0, 3.0])*

AddResponseSpectrumLoadEx(*rsaCode: int, rsaCombination: int, varSet1Names: list, varSet1Vals: list, varSet2Names: list, varSet2Vals: list, varDataPairs: list*) [source]

Adds Response Spectrum load item to the currently active load case.

Parameters:

- **rsaCode** (*int*) – Response Spectrum Loading Code. Refer to the following table for the integers corresponding to different codes.
- **rsaCombination** (*int*) – Modal combination rule. (SRSS = 0, ABS = 1, CQC = 2, ASCE = 3, TEN = 4, CSM = 5, GRP = 6)
- **varSet1Names** (*list of string*) – List of string containing parameter key words. Refer to the Technical Reference sections as indicated below.
- **varSet1Vals** (*list of float*) – List of Parameters values corresponding to the keywords supplied in varSet1Names array.
- **varSet2Names** (*list of string*) – List of string containing parameter key words for the spectrum generation data command. NULL can be used if not needed.
- **varSet2Vals** (*list of float*) – List of Parameters values corresponding to the keywords supplied in varSet1Names array. NULL can be used if not needed.
- **varDataPairs** (*list of float*) – List of containing pairs of time period and acceleration data. NULL can be used if not needed. Inputs (varSet2Names, varSet2Vals) and (varDataPairs) are mutually exclusive, i.e. if one is specified, other should not specified
- **Notes**
- **sections** (- *Techincal Reference*) –

nTypeNo	Seismic Code	Parameters	Ref. Sec.
0	Generic or Custom	DEC, ECC, X, Y, Z, ACC, DIS, SCA, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, FF1, FF2, DOM, SIG, SAV, IMR, STA	TR.32.10.1.1
1	IS:1893 Part 1 2002	TOR, DEC, ECC, X, Y, Z, ACC, DIS, SCA, DAM, CDA, MDA, MIS, ZPA, IGN, DOM, SIG, SAV, IMR, STA SOI, CHE, RF	TR.32.10.1.7
2	IS:1893 2016	TOR, DEC, ECC, X, Y, Z, ACC, DIS, SCA, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, IGN, DOM, SIG, SAV, IMR, STA SOI, CHE, RF	TR.32.10.1.8
4	ENV 1998-1:1994	ELA, DES, X, Y, Z, ACC, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, DOM, SIG, SAV, IMR, STA SOI, ALP, Q	TR.32.10.1.4
5	EN 1998-1:2004	ELA, DES, RS1, RS2, X, Y, Z, ACC, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, DOM, SIG, SAV, IMR, STA SOI, ALP, Q	TR.32.10.1.5
6	IBC 2006	X, Y, Z, ACC, DAM, CDA, MDA, LIN, LOG, MISC, ZPA, DOM, SIG, SAV IMR, STA ZIP, LAT, LON, SS, S1, SCA, FA, FV, TL	TR.32.10.1.10
7	IBC 2012	X, Y, Z, ACC, DAM, CDA, MDA, LIN, LOG, MISC, ZPA, DOM, SIG, SAV IMR, STA ZIP, LAT, LON, SS, S1, SCA, FA, FV, TL	TR.32.10.1.11
8	IBC 2015	X, Y, Z, ACC, DAM, CDA, MDA, LIN, LOG, MISC, ZPA, DOM, SIG, SAV IMR, STA ZIP, LAT, LON, SS, S1, SCA, FA, FV, TL	TR.32.10.1.12
10	SNiP II-7-81	A, X, KWX, KX1, Y, KWY, KY1, Z, KWZ, KZ1, ACC, SCA, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, DOM, SIG, SOI, SAV	TR.32.10.1.14
11	SP 14.13330.2011	ECC, A, X, Y, Z, ACC, SCA, DAM, LOG, MIS, ZPA, DOM, SIG, SOI	TR.32.10.1.15
12	CANADIAN: NRC-2005	TOR, DEC, ECC, X, Y, Z, ACC, DIS, SCA, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, DOM, SIG, SAV, IMR, STA	TR.32.10.1.2
13	CANADIAN: NRC-2010	TOR, DEC, ECC, X, Y, Z, ACC, DIS, SCA, DAM, CDA, MDA, LIN, LOG, MIS, ZPA, DOM, SIG, SAV, IMR, STA	TR.32.10.1.3
14	GB 50011 2010	X, Y, Z, ALP, DAM, CDA, MDA, LIN, LOG, MISS, ZPA, DOM, SIG, INT, FRE, FOR,RAR, GRO, SCL	TR.32.10.1.6

- **code** (- *Following values should be specified for INT parameter of GB 50011 2010*) –

Fortification Intensity	Value
6	0
7	1
7A	2
8	3
8A	4
9	5

Returns:

Returns TRUE if successful Returns FALSE if unsuccessful

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddResponseSpectrumLoadEx(0, 0, ['DEC', 'ECC'], [1.0, 2.0], [], [], [])
```

`AddSeismicDefElementWeight(pressure: float, elementList: list)` [source]

Adds a pressure to Seismic Definition.

Parameters:

- **pressure** (*float*) – Pressure Value
- **elementList** (*List of int*) – 'List of element ID list.

Returns:

- *int* – True if successful. False if unsucessful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefElementWeight(1.0, [1, 2, 3])*

`AddSeismicDefFloorWeight(rangeType: int, loadDirection: int, pressure: float, grpOrOneWay: int, yMIN: float, yMAX: float, zMIN: float, zMAX: float, xMIN: float, xMAX: float)` [source]

Adds a floor weight to Seismic Definition.

Parameters:

- **rangeType** (*int*) –

  Type of the Range :

Value	Range Type
0	X-RANGE
1	Y-RANGE
2	Z-RANGE
3	Group Load

- **loadDirection** (*int*) –

  Load direction :

Value	Direction
0	Global X
1	Global Y
2	Global Z

- **pressure** (*float*) – Magnitude of the pressure or concentrate load on the elemen
- **grpOrOneWay** (*int*) – One-Way Load (if it is either "" or "0") or corresponding group name to add Floor Group Load (if it contains Group string name). Notes: - Group name should be of FLOOR group type.
- **yMIN** (*float*) – Y range from which the load start(in global coordinate).
- **yMAX** (*float*) – Y range at which the load end(in global coordinate).
- **zMIN** (*float*) – Z range from which the load start(in global coordinate).
- **zMAX** (*float*) – Z range at which the load end(in global coordinate).
- **xMIN** (*float*) – X range from which the load start(in global coordinate).

- **xMAX** (*float*) – X range at which the load end(in global coordinate).

Returns:

- *int* – True if successful. False if unsucessful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefFloorWeight(0, 1, 2.0, 3, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0)*

AddSeismicDefJointWeight(*weight: float, nodeList: list*) [source]

> Adds joint self weight to Seismic Definition.

Parameters:

- **weight** (*float*) – Weight value.
- **nodeList** (*list of int*) – List of Node number IDs

Returns:

- *int* – Returns 0 if OK . Returns -1 if General error. Returns -100 if Invalid argument. Returns -106 if 1 dimensional array of long expected. Returns -113 if Integer array/Integer expected. Returns -2006 if Invalid Node Number. Returns -8034 if Seisemic Code not found.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefJointWeight(1.0, [1, 2, 3])*

AddSeismicDefMemberWeight(*varSeismicType: int, loadType: int, weight: float, startDist: float, endDist: float, memberList: list*) [source]

Adds member concentrated/uniform weight to Seismic Definition.

Parameters:

- **varSeismicType** (*int*) –

  Type of seismic code:

Value	Seismic Code
0	AUTO DETECT
1	ALGERIAN: RPA
2	CANADIAN: NRC-1995
3	CANADIAN: NRC-2005
4	CANADIAN: NRC-2010
5	CANADIAN: NRC-2020
6	Chinese: GB50011-2001
7	Chinese: GB50011-2010
8	COLOMBIAN: NSR 95
9	COLOMBIAN: NSR 2010
10	IBC 2000
11	IBC 2003 ASCE 7-02
12	IBC 2006/2009 ASCE 7-05
13	IBC 2012 ASCE 7-10
14	IBC 2015 ASCE 7-10
15	IBC 2018 ASCE 7-16
16	Indian: IS 1893-1984
17	Indian: IS 1893-2002/2005
18	Indian: IS 1893-2016
19	Indian: IS 1893(Part4) 2015
20	JAPANESE (AIJ)
21	MEX: CFE-1993
22	MEX: NTC-1987
23	TURKISH
24	UBC 1985
25	UBC 1994
26	UBC 1997

- **loadType** (*int*) – 1 for uniform loadType and 2 for concentrated loadType
- **weight** (*float*) – Uniform weight.
- **startDist** (*float*) – Starting distance(= distance from member starting node to weight starting point)
- **endDist** (*float*) – Ending distance(= distance from member starting node to weight ending point)
- **memberList** (*list of int*) – List of Member ID to add member concentrated/uniform weight

Returns:

- *bool* – True if successful adds member concentrated/uniform weight to Seismic Definition. False if unsucessful
- *>>> from openstaadpy import os_analytical*

- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefMemberWeight(1, 2, 3.0, 4.0, 5.0, [6, 7, 8])*

AddSeismicDefSelfWeight(*varWeightFactor: float*) [source]

Adds self weight to Seismic Definition.

Parameters:

varWeightFactor (*float*) – Weight Factor to add to self weight

Returns:

- *bool* – True if successful. False if unsucessful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefSelfWeight(1.0)*

AddSeismicDefWallArea(*nTypeNo: int, direction: str, sizeArray: list*) [source]

Adds wall area to Seismic Definition. .. note:: - Wall Area is only available in IS1893-2016 seismic code.

Parameters:

- **nTypeNo** (*int*) –

  Type of seismic code:

Value	Seismic Code
15	Indian: IS 1893-2016

- **direction** (*string*) – Direction value. [X directin or Z direction]
- **sizeArray** (*List of float*) – Length and Width list consisting of consecutive length and width measurements

Returns:

- *int* – Returns 0 if OK . Returns -1 if General error. Returns -107 if 1 dimensional array of long expected. Returns -8034 if Invalid seismic code. Returns -8038 if Invalid Direction.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicDefWallArea(15, "X", [10.0, 20.0])*

AddSeismicDefinition(*varType: int, varAccidental: int*) [source]

Adds a Seismic Definition with default parameters.

Parameters:

- **varType** (*int*) –

  Type of seismic code:

Value	Seismic Code
0	UBC 1985
1	UBC 1994
2	UBC 1997
3	Indian: IS 1893-1984
4	Indian: IS 1893-2002/2005
5	IBC 2000
6	IBC 2003
7	COLOMBIAN: NSR 98
8	JAPANESE (AIJ)
9	ALGERIAN: RPA
10	MEX: CFE-1993
11	MEX: NTC-1987
12	Indian: IS 1893-2016
13	Indian: IS 1893(Part4) 2015
14	IBC 2006
15	IBC 2012
16	IBC 2015
17	IBC 2018
18	CANADIAN: NRC-2005
19	CANADIAN: NRC-2010
20	CANADIAN: NRC-1995
21	COLOMBIAN: NSR 2010
22	Chinese: GB50011-2001
23	Chinese: GB50011-2010
24	TURKISH

- **varAccidental** (*int*) – '1' to consider accidental torsion else '0' ignore

Returns:

True if successful. False if unsucessful

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddSeismicDefinition(1, 0)
```

AddSeismicLoad(*loadDirection: int, factor: float*) [source]

Adds a Seismic Definition with default parameters.

Parameters:

- **loadDirection** (*int*) – Load direction: (= 0 to 2 for global X, Y and Z, respectively).
- **factor** (*float*) – 'Multiplication factor to be used to multiply the seismic load.

Returns:

- *int* – Returns 0 if OK. Returns -1 if General error. Returns -8001 if Load direction is invalid.
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddSeismicLoad(0, 1.0)*

AddSelfWeightInXYZ(*varInDirection: int, varLoadFactor: float*) [source]

Adds self-weight to the active load case for all entities (beams, plates, solids).

Parameters:

- **varInDirection** (*int*) – Direction index for self-weight (1 = X, 2 = Y, 3 = Z).
- **varLoadFactor** (*float*) – Multiplying factor for self-weight.

Returns:

True if self-weight was added successfully. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddSelfWeightInXYZ(2, 1.0)
>>> print(result)
```

AddSelfWeightInXYZToGeometry(*varGeomNumberIDs: list, varInDirection: int, varLoadFactor: float*) [source]

Adds self-weight to specified geometry entities in the active load case.

Parameters:

- **varGeomNumberIDs** (*list of int*) – List of beam, plate, or solid number IDs.
- **varInDirection** (*int*) – Direction index for self-weight (1 = X, 2 = Y, 3 = Z).
- **varLoadFactor** (*float*) – Multiplying factor for self-weight.

Returns:

True if self-weight was added to the specified geometries. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddSelfWeightInXYZToGeometry([1,2], 3, 0.9)
>>> print(result)
```

AddStrainLoad(*elementIds: list, varAxialElong: float*) [source]

Adds STRAIN LOAD to beam or plate elements.

Parameters:

- **elementIds** (*list of int*) – List of element IDs.
- **varAxialElong** (*float*) – Initial axial elongation (+)/ shrinkage (-) in member due to misfit, etc.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddStrainLoad([1], 0.001)
```

AddSupportDisplacement(*nodeIds: list, varDirection: int, varDispValue: float*) [source]

Adds support displacement to one or more nodes.

Parameters:

- **nodeIds** (*list of int*) – List of node IDs.
- **varDirection** (*int*) – Direction index (1 = X, 2 = Y, 3 = Z).
- **varDispValue** (*float*) – Displacement value in the specified direction.

Returns:

True if the support displacement was added successfully. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.AddSupportDisplacement([1], 1, 5.0)
>>> print(result)
```

AddTemperatureLoad(*elementIds: list, varTempAxialElong: float, varTempDiffTopAndBtm: float, varTemDiffSide: float*) [source]

Adds TEMPERATURE LOAD to beam or plate elements.

Parameters:

- **elementIds** (*list of int*) – List of element IDs.
- **varTempAxialElong** (*float*) – Change in temperature.
- **varTempDiffTopAndBtm** (*float*) – Temperature difference from the top to the bottom of the element (for calculating bending).
- **varTemDiffSide** (*float*) – Temperature difference from side to side of the element (local Z axis).

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddTemperatureLoad([1], 20.0, 30.0, 40.0)
```

AddWindDefinition(*varTypeNo: int, varTypeName: str*) [source]

Adds a Wind Definition named “varTypeName” with number ID varTypeNo.

Parameters:

- **varTypeNo** (*int*) – Wind Definition Type number ID.
- **varTypeName** (*string*) – String name of this new type.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddWindDefinition(1, "Wind Load 1")
```

AddWindDefinitionASCE7Parameters(*varTypeNo: int, code: int, windSpeed: float, heightAboveSeaLvl: float, bldgclass: int, bldgtype: int, expCat: int, varEscarpment: bool, wallType: int, varIsFlexible: bool, varEscarpmentData: list, varbldgData: list, varUnitsData: list, varFactorsUserInput: list, varFactors: list*) [source]

Generates the Wind Definition Parameters using ASCE CODE.

Parameters:

- **varTypeNo** (*int*) – Wind Definition Type number ID
- **code** (*int*) –

Value
0
1
2
3

- **windSpeed** (*float*) – Wind speed.
- **heightAboveSeaLvl** (*float*) – Ground Height above sea level. [Required only for ASCE7-2016. For other versions give 0.0]
- **bldgclass** (*int*) –

value	Building
0	Category
1	Category
2	Category
3	Category

- **bldgtype** (*int*) –

Value	Building Typ
0	Building Struc
1	Chimney, Tan
2	Solid Signs
3	Open Signs
4	Lattice Frame
5	Trussed Towe

- **expCat** (*int*) –

Value
0
1
2
3

- **varEscarpment** (*bool*) – Consider Wind Speed-up over Hills (FALSE) or Escarpment (TRUE).
- **wallType** (*int*) –

Value
0
1
2

- **varIsFlexible** (*bool*) – Consider structure is Flexible (TRUE) or RIGID (FALSE).
- **varEscarpmentData** (*List of float*) – Float list of size 4 containing information describing Hills or Escarpment +——-+——————
- **varbldgData** (*List of float*) –

  Float list of size 7 containing information describing the building based on structure type :

  - **Building Data :**

Index	Item
0	**Enclosure Classification :** ▪ Before 2016 : Open Building (0)/ Partially Enclosed (1)/ Enclos ▪ [2016] : Open Building (0)/ Partially Open (1)/ Partially Enclos
1	Building Height
2	Building length long the direction of Wind (L)
3	Building length normal to the direction of Wind (B)
4	Building Natural Frequency
5	Building Damping Ratio

  - **Tank Data :**

Index	Item
0	**Horizontal Cross-section Type :** ▪ Before 2016 : Square (0)/ Square Diagonal (1)/ Hexagonal or Octagonal (2)/ Round (3 ▪ [2016] : Square (0)/ Square Diagonal (1)/ Hexagonal (2)/ Octagonal Non-axisymmetri
1	Tank Height
2	Least Horizontal Dimension (W)
3	Depth of producing elements like Spoilers and Ribs (D′)
4	Structure Natural Frequency
5	Structure Damping Ratio

- **Solid Sign Data :**

Index
0
1
2
3
4

- **Open Sign/Lattice Framework Data :**

Index	Item
0	Orienta
1	Height
2	Width
3	Diame
4	Ratio c
5	Structu
6	Structu

- **Trussed Tower Data :**

Index	Item
0	Horizontal Cross Sectio Type: Tri
1	Tank Height (H)
2	Width
3	Ratio of Solid Area to Gross Are
4	Structure Natural Frequency
5	Structure Damping Ratio

- **varUnitsData** (*List of int*) – Float list of size 7 containing Units of data inputs +——-+————————————————————————
+=======+===
| 0 | Unit of Wind Speed {mph(VelocityUnit::mph or 0) or m/sec(VelocityUnit::metersec or 1) or cm/sec(VelocityUnit::cmsec or 2) or mm
{inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4)
——
——
——
+———————————————————————————————————————
Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)} | +——-+————————————————
m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd o
or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm o
- **varFactorsUserInput** (*List of int*) – Float list of size 7 containing information describing whether Factors are User Input or Calculated Calculated(0) | +——-+——————————————————————————+ | 4 | Ke is User Input(1) or Calculated(0) [Required only
- **varFactors** (*List of int*) – Float list of size 7 containing information describing whether Factors are User Input or Calculated +——-+——————————————————————+ | 7 | Factor Gcpi | +——-+——————————————————+

Returns:

- *bool* – Returns True if succesful
- *>>> from openstaadpy import os_analytical*
- *>>> staad_obj = os_analytical.connect()*
- *>>> staad_obj.Load.AddWindDefinitionASCE7Parameters(1, "ASCE7-10", 100.0, 10.0, 1, 1, 1, 1, [1.0, 2.0, 3.0], [1.0, 2.0, 3.0])*

`AddWindExposure(varTypeNo: int, varExposureFactor: float, varNodeArray: list)` [source]

Adds Wind Exposures factor to Wind Definitions and assign to nodes.

Parameters:

- **varTypeNo** (*int*) – Wind Definition Type number ID.
- **varExposureFactor** (*float*) – Exposure factor.
- **varNodeArray** (*list of int*) – Node number ID list.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddWindExposure(1, 1.0, [1,2,3])
```

`AddWindIntensity(varTypeNo: int, varIntensity: list, varHeight: list)` [source]

Adds to Wind Definitions Wind Intensity by giving Intensity vs. Height.

Parameters:

- **varTypeNo** (*int*) – Wind Definition Type number ID.
- **varIntensity** (*list of float*) – Intensity values float list
- **varHeight** (*list of float*) – Height value float list.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddWindIntensity(1,[5.2], [10.0])
```

AddWindLoad(*varTypeNo: int, varDirection: int, dFraction: float, varOpenStructure: int, dYMIN: float, dYMAX: float, dZMIN: float, dZMAX: float, dXMIN: float, dXMAX: float*) [source]

Adds a wind load.

Parameters:

- **varTypeNo** (*int*) – Wind Definition Type number ID.
- **varDirection** (*int*) –

  Wind load direction:

Value	Direction
1	Global X
3	Global Z
4	Global -X
6	Global -Z

- **dFraction** (*float*) – Factor to be used to multiply the wind loads. Negative signs may be used to indicate opposite direction of resulting load (default=1.0).
- **varOpenStructure** (*int*) – For Open-type of structure enter 1 , closed-type of structure 0
- **dYMIN** (*float*) – Ymin of GLOBAL Y range in which Wind load applied (assume Y axis is vertical).
- **dYMAX** (*float*) – Ymax of GLOBAL Y range in which Wind load applied (assume Y axis is vertical).
- **dZMIN** (*float*) – Zmin of GLOBAL Z range in which Wind load applied (assume Y axis is vertical).
- **dZMAX** (*float*) – Zmax of GLOBAL Z range in which Wind load applied (assume Y axis is vertical).
- **dXMIN** (*float*) – Xmin of GLOBAL X range in which Wind load applied (assume Y axis is vertical).
- **dXMAX** (*float*) – Xmax of GLOBAL X range in which Wind load applied (assume Y axis is vertical).

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.AddWindLoad(1, 3, 1.0, 1, 0.0, 10.0, 0.0, 10.0, 0.0, 10.0)
```

BeginLoadMerging() [source]

Begin Load Merging

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.BeginLoadMerging()
```

`ClearPrimaryLoadCase(varLoadCaseNos: list, isReferenceLoad: bool)` [source]

Clears the load items in a specified Primary Load cases or Reference Load cases.

Parameters:

- **varLoadCaseNos** (*list*) – Primary load case reference ID(s) list.
- **isReferenceLoad** (*bool*) – If reference load case(s): True or False.

Returns:

Returns 1 if OK Returns 0 if failed to delete load(s)

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.ClearPrimaryLoadCase([1, 2, 3], False)
```

`ClearReferenceLoadCase(varLoadCaseNos: list)` [source]

Clears the load items in a specified Primary Load cases or Reference Load cases.

Parameters:

varLoadCaseNos (*list*) – Primary load case reference ID(s) list.

Returns:

Returns 1 if OK Returns 0 if failed to delete load(s)

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.ClearReferenceLoadCase([1, 2, 3])
```

`ComputeWallWindPressureProfile(loadingCode: int, windSpeed: float, bldgClass: int, bldgType: int, expCat: int, bEscarpment: bool, varUnitsData: list, varEscarpmentData: list, varBldgData: list, wallType: int)` [source]

Generates the wall wind pressure profile using ASCE CODE.

Parameters:

- **loadingCode** (*int*) –

  ASCE CODE:

Value	ASCE CODE
ASCE7Y95 = 0	ACSE 7-1995
ACSE702 = 1	ACSE 7-2002
ACSE705_10 = 2	ACSE 7-2010

- **windSpeed** (*float*) – Wind speed. Default value 85 mph.
- **bldgClass** (*int*) –

  Building Classification Category:

Value	Building Classification Category
TypeI = 0	Category I
TypeII = 1	Category II
TypeIII = 2	Category III
TypeIV = 3	Category IV

- **bldgtype** (*int*) –

  Structure Type:

Value	Structure Type
Building = 0	Building Structures
Chimney = 1	Chimney, Tank and similar structures
Solidsign = 2	Solid Signs
Opensign = 3	Open Signs
Laticeframe = 4	Lattice Framework
Trusstower = 5	Trussed Tower

- **expCat** (*int*) –

  Exposure Category:

Value	Exposure Category
ExpA = 0	Exposure A
ExpB = 1	Exposure B
ExpC = 2	Exposure C
ExpD = 3	Exposure D

- **bEscarpment** (*bool*) – Consider Wind Speed-up over Hills (FALSE) or Escarpment (TRUE).
- **varUnitsData** (*list of long*) –

  Integer list of size 8 containing Units of data inputs:

Index	Data
0	Unit of Wind Speed {mph(VelocityUnit::mph or 0) or m/sec(VelocityUnit::metersec or 1) or cm/sec(VelocityUnit::cmsec or 2) or mm/sec(VelocityUnit::mmsec or 3) or kmph(VelocityUnit::kmph or 4) or in/sec(VelocityUnit::inchsec or 5) or ft/sec(VelocityUnit::ftsec or 6) or Yd/sec(VelocityUnit::yardsec or 7)}
1	Unit of Ground height above sea level {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
2	[Escarpment] Unit of Height (H) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
3	[Escarpment] Unit of Distance upwind of crest (Lh) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
4	[Escarpment] Unit of Distance from the crest to the building (x) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
5	[Building]Unit of Height/ [Tank]Unit of Height/ [Solid Sign]Unit of Height/ [Open Sign/Lattice]Unit of Height/ [Trusses Tower]Unit of Height {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
6	[Building]Unit of Length/ [Tank]Unit of Width/ [Solid Sign]Unit of M dimension/ [Open Sign/Lattice]Unit of Width/ [Trusses Tower]Unit of Width {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
7	[Building]Unit of Width/ [Tank]Unit of Depth/ [Solid Sign]Unit of N dimension/ [Open Sign/Lattice]Unit of Diameter/ [Trusses Tower]Not Applicable {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}

- **varescarpmentData** (*list*) –

  Information describing Hills or Escarpment:

Index	Data
0	Type: 2D Ridge (0), 2D Escarpment (1), 3D Escarpment (2)
1	Height (H)
2	Distance upwind of crest (Lh)
3	Distance from the crest to the building (x)

- **varbldgData** (*list*) –

List of size 7 containing information describing the building based on structure type.

- **Building Data :**

Index	Item
0	Enclosure Classification: Open Building (0)/ Partially Enclosed (1)/ Enclosed Building (2)
1	Building Height
2	Building length long the direction of Wind (L)
3	Building length normal to the direction of Wind (B)
4	Building Natural Frequency
5	Building Damping Ratio

- **OR Tank Data :**

Index	Item
0	Horizontal Cross-section Type:- Square (0)/ Square Diagonal (1)/ Hexagonal or Octagonal (2)/ Round (3)
1	Tank Height (H)
2	Least Horizontal Dimension (W)
3	Depth of producing elements like Spoilers and Ribs (D’)
4	Structure Natural Frequency
5	Structure Damping Ratio

- **OR Solid Sign Data :**

Index	Item
0	Solid Sign Height (H)
1	Solid Sign M Dimension (M)
2	Solid Sign N Dimension (N)
3	Structure Natural Frequency
4	Structure Damping Ratio

- **OR Open Sign/Lattice Framework Data :**

Index	Item
0	Orientation Type: Flat (0)/ Rounded (1)
1	Height (H)
2	Width
3	Diameter of typical round member
4	Structure Natural Frequency
5	Structure Damping Ratio
6	Ratio of Solid Area to Gross Area

- **OR Trussed Tower Data :**

Index	Item
0	Horizontal Cross Sectio Type: Triangle (0)/ Square (1)
1	Height (H)
2	Width
3	Structure Natural Frequency
4	Structure Damping Ratio
5	Ratio of Solid Area to Gross Area(in percetage)

- **wallType** (*int*) –

  Building wall to generate Wind Load on:

Value	Wall Type
WindWard = 0	WindWard
LeeWard = 1	Leeward
SideWall = 2	SideWall

- (0 to 2 for WindWard, Leeward and SideWall, respectively).

Returns:

Returns number of Height or Intensity data.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.ComputeWallWindPressureProfile(2, 90.0, 1, 0, 2, False, [0]*8, [0.0]*4, [0.0]*7, 0)
```

ComputeWallWindPressureProfileASCE72016(*windSpeed: float, heightAboveSeaLvl: float, bldgClass: int, bldgType: int, expCat: int, bEscarpment: bool, varUnitsData: list, varEscarpmentData: list, varBldgData: list, wallType: int*) [source]

Modifies or adds a seismic parameter in the existing seismic definition.

Parameters:

- **windSpeed** (*float*) – Wind speed. Default value 85 mph.
- **heightAboveSeaLvl** (*float*) – Ground height above sea level. Used only for ASCE7-2016 Wind. Default value 0.0 ft.
- **bldgClass** (*int*) –

  Building Classification Category:

Value	Building Classification Category
TypeI = 0	Category I
TypeII = 1	Category II
TypeIII = 2	Category III
TypeIV = 3	Category IV

- **bldgtype** (*int*) –

  Structure Type:

Value	Structure Type
Building = 0	Building Structures
Chimney = 1	Chimney, Tank and similar structures
Solidsign = 2	Solid Signs
Opensign = 3	Open Signs
Laticeframe = 4	Lattice Framework
Trusstower = 5	Trussed Tower

- **expCat** (*int*) –

  Exposure Category:

Value	Exposure Category
ExpA = 0	Exposure A
ExpB = 1	Exposure B
ExpC = 2	Exposure C
ExpD = 3	Exposure D

- **bEscarpment** (*bool*) – Consider Wind Speed-up over Hills (FALSE) or Escarpment (TRUE).
- **varUnitsData** (*list of long*) –

  Integer list of size 8 containing Units of data inputs:

Index	Data
0	Unit of Wind Speed {mph(VelocityUnit::mph or 0) or m/sec(VelocityUnit::metersec or 1) or cm/sec(VelocityUnit::cmsec or 2) or mm/sec(VelocityUnit::mmsec or 3) or kmph(VelocityUnit::kmph or 4) or in/sec(VelocityUnit::inchsec or 5) or ft/sec(VelocityUnit::ftsec or 6) or Yd/sec(VelocityUnit::yardsec or 7)}
1	Unit of Ground height above sea level {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
2	[Escarpment] Unit of Height (H) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
3	[Escarpment] Unit of Distance upwind of crest (Lh) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
4	[Escarpment] Unit of Distance from the crest to the building (x) {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
5	[Building]Unit of Height/ [Tank]Unit of Height/ [Solid Sign]Unit of Height/ [Open Sign/Lattice]Unit of Height/ [Trusses Tower]Unit of Height {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
6	[Building]Unit of Length/ [Tank]Unit of Width/ [Solid Sign]Unit of M dimension/ [Open Sign/Lattice]Unit of Width/ [Trusses Tower]Unit of Width {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}
7	[Building]Unit of Width/ [Tank]Unit of Depth/ [Solid Sign]Unit of N dimension/ [Open Sign/Lattice]Unit of Diameter/ [Trusses Tower]Not Applicable {inch(LengthUnit::In or 0) or ft(LengthUnit::Ft or 1) or foot(LengthUnit::foot or 2) or cm(LengthUnit::cm or 3) or m(LengthUnit::M or 4) or mm(LengthUnit::Mm or 5) or dm(LengthUnit::Dm or 6) or Km(LengthUnit::Km or 7) or yard(LengthUnit::Yd or 8) or mile(LengthUnit::mil or 9)}

- **varescarpmentData** (*list*) –

  Information describing Hills or Escarpment:

Index	Data
0	Type: 2D Ridge (0), 2D Escarpment (1), 3D Escarpment (2)
1	Height (H)
2	Distance upwind of crest (Lh)
3	Distance from the crest to the building (x)

- **varbldgData** (*list*) –

List of size 7 containing information describing the building based on structure type.

- **Building Data :**

Index	Item
0	Enclosure Classification: Open Building (0)/ Partially Enclosed (1)/ Enclosed Building (2)
1	Building Height
2	Building length long the direction of Wind (L)
3	Building length normal to the direction of Wind (B)
4	Building Natural Frequency
5	Building Damping Ratio

- **OR Tank Data :**

Index	Item
0	Horizontal Cross-section Type:- Square (0)/ Square Diagonal (1)/ Hexagonal or Octagonal (2)/ Round (3)
1	Tank Height (H)
2	Least Horizontal Dimension (W)
3	Depth of producing elements like Spoilers and Ribs (D’)
4	Structure Natural Frequency
5	Structure Damping Ratio

- **OR Solid Sign Data :**

Index	Item
0	Solid Sign Height (H)
1	Solid Sign M Dimension (M)
2	Solid Sign N Dimension (N)
3	Structure Natural Frequency
4	Structure Damping Ratio

- **OR Open Sign/Lattice Framework Data :**

Index	Item
0	Orientation Type: Flat (0)/ Rounded (1)
1	Height (H)
2	Width
3	Diameter of typical round member
4	Structure Natural Frequency
5	Structure Damping Ratio
6	Ratio of Solid Area to Gross Area

- **OR Trussed Tower Data :**

Index	Item
0	Horizontal Cross Sectio Type: Triangle (0)/ Square (1)
1	Height (H)
2	Width
3	Structure Natural Frequency
4	Structure Damping Ratio
5	Ratio of Solid Area to Gross Area(in percetage)

- **wallType** (*int*) –

  Building wall to generate Wind Load on:

Value	Wall Type
WindWard = 0	WindWard
LeeWard = 1	Leeward
SideWall = 2	SideWall

- (0 to 2 for WindWard, Leeward and SideWall, respectively).

Returns:

Returns number of Height or Intensity data.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.ComputeWallWindPressureProfileASCE72016(100, 10, 1, 1, 1, 1, [1], [1.0], [1.0], 0)
```

CreateLoadEnvelop(*envelopNumber: int, envelopType: int, loadCaseList: list*) [source]

Creates a Load Envelop with specified primary load case(s) and envelop type.

Parameters:

- **envelopNumber** (*int*) – Load Envelop reference ID

- **envelopType** (*int*) –

  Type of the load envelop:

Value	Load Envelop Type
0	NONE
1	STRESS
2	SERVICEABILITY
3	COLUMN
4	CONNECTION
5	STRENGTH
6	TEMPORARY

- **loadCaseList** (*list of int*) – Load Case IDs for which to create a load envelop

Returns:

True OK. False General error.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.CreateLoadEnvelop(1, 1, [1,2,3])
>>> print(result)
```

CreateLoadList(*listType: int, loadCaseList: list[int]*) [source]

Creates a load list.

Parameters:

- **listType** (*int*) – Load list type: 0 and 1 for load list and load envelope list, respectively.
- **loadCaseList** (*list of int*) – Load Case reference IDs for which to create a load envelop

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.CreateLoadList(0, [1,2])
>>> print(result)
```

CreateNewLoadCombination(*loadCombTitle: str, loadCombNo: int*) [source]

Creates a new load combination case.

Parameters:

- **loadCombTitle** (*str*) – Title of the load combination.
- **loadCombNo** (*int*) – Load combination number.

Returns:

Load number ID assigned to the load combination. Returns -1 in case of an error. Returns -8004 if it fails to create the load.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> comb_id = staad_obj.Load.CreateNewLoadCombination("DL+LL", 2)
>>> print(comb_id)
```

CreateNewPrimaryLoad(*primaryLoadTitle: str*) [source]

Creates a new PRIMARY load case.

Parameters:

primaryLoadTitle (*str*) – Title of the primary load case.

Returns:

Load number ID if the load case is created successfully. Returns -1 in case of a general error. Returns -8004 if the load case creation fails specifically.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> load_id = staad_obj.load.CreateNewPrimaryLoad("Dead Load")
>>> print(load_id)
```

CreateNewPrimaryLoadEx(*primaryLoadTitle: str, loadType: int*) [source]

Creates new PRIMARY load case.

Parameters:

- **primaryLoadTitle** (*string*) – The primary load case string title.
- **loadType** (*int*) –

  Type of the load:

Value	Load Type	Value	Load Type
0	Dead	12	Traffic
1	Live	13	Temp
2	Roof Live	14	Imperfection
3	Wind	15	Accidental
4	Seismic-H	16	Flood
5	Seismic-V	17	Ice
6	Snow	18	Wind Ice
7	Fluids	19	Crane Hook
8	Soil	20	Mass
9	Rain	21	Gravity
10	Ponding	22	Push
11	Dust	23	None

Returns:

Returns load Number of newly created Primary load Case. Returns -1 if general error. Returns -8004 if fail to create load.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> load_id = staad_obj.Load.CreateNewPrimaryLoadEx("Live Load", 1)
>>> print(load_id)
```

`CreateNewPrimaryLoadEx2(primaryLoadTitle: str, LoadType: int, LoadCaseNo: int)` [source]

Creates new PRIMARY load case.

Parameters:

- **primaryLoadTitle** (*string*) – The primary load case string title.
- **loadType** (*int*) –

  Type of the load:

Value	Load Type	Value	Load Type
0	Dead	12	Traffic
1	Live	13	Temp
2	Roof Live	14	Imperfection
3	Wind	15	Accidental
4	Seismic-H	16	Flood
5	Seismic-V	17	Ice
6	Snow	18	Wind Ice
7	Fluids	19	Crane Hook
8	Soil	20	Mass
9	Rain	21	Gravity
10	Ponding	22	Push
11	Dust	23	None

- **loadCaseNo** (*int*) – The load case number.

Returns:

Returns load Case number of newly created Primary load Case. Returns 0 if not successfully

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> load_id = staad_obj.Load.CreateNewPrimaryLoadEx2("Wind", 3, 5)
>>> print(load_id)
```

CreateNewReferenceLoad(*nodeNo: int*, *referenceLoadCaseTitle: str*, *loadType: int*) [source]

Creates a new reference load case.

Parameters:

- **nodeNo** (*int*) – Reference ID to be assigned to the new reference load case.
- **referenceLoadCaseTitle** (*str*) – Title of the reference load case.
- **loadType** (*int*) – Type of load.

Returns:

Reference load case number ID. Returns -1 in case of an error. Returns -8004 if it fails to create the load.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> ref_id = staad_obj.Load.CreateNewReferenceLoad(1, "Ref Load", 0)
>>> print(ref_id)
```

DeleteDirectAnalysisDefinition()

[source]

Deletes whole Direct Analysis Definition.

Returns:

Returns True if successful. Returns False if unsuccessful.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteDirectAnalysisDefinition()
```

DeleteDirectAnalysisDefinitionParameter(*pParamType: int*)

[source]

Deletes respective parameters from Direct Analysis Definition based on the Parameter Type passed as argument (FLEX/AXIAL).

Parameters:

pParamType (*int*) –

Integer indicating type of direct analysis parameter to be added. Integer value should be taken from following table :

Value	AnalysisCommand
FLEX = 0	DirectAnalysisParameterTypes.FLEX
AXIAL = 2	DirectAnalysisParameterTypes.AXIAL

Returns:

Returns True if successful. Returns False if unsuccessful.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteDirectAnalysisDefinitionParameter(0)
```

DeleteLoadEnvelop(*varEnvNo: int*)

[source]

Deletes a specified load envelop.

Parameters:

varEnvNo (*int*) – Load Envelop reference ID.

Returns:

Returns 0 if OK Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteLoadEnvelop(1)
```

`DeleteLoadList(varLoadListIndex: int)` [source]

Deletes specified load list.

Parameters:

varLoadListIndex (*int*) – Load list index.

Returns:

Returns 0 if OK Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteLoadList(1)
```

`DeletePrimaryLoadCases(varLoadCaseNos: list, varIsReferenceLoads: bool)` [source]

Deletes specified Primary/Reference Load Cases.

Parameters:

- **varLoadCaseNos** (*List of int*) – List of Primary/Reference load case reference ID.
- **varIsReferenceLoads** (*bool*) – If reference load case(s): TRUE or FALSE

Returns:

Returns 0 if OK Returns -1 if failed to delete load(s)

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeletePrimaryLoadCases([1, 2, 3], False)
```

`DeleteReferenceLoadCases(varLoadCaseNos: list)` [source]

Deletes specified Reference Load Cases.

Parameters:

varLoadCaseNos (*List of int*) – List of Reference load case reference ID.

Returns:

Returns 0 if OK Returns -1 if failed to delete load(s)

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteReferenceLoadCases([1, 2, 3])
```

DeleteWindDefinition(*nTypeNo: int*) [source]

Deletes Wind definition. All defintions will be deleted if this input is set as 0.

Parameters:

nTypeNo (*int*) – Type of Wind.

Returns:

Returns 0 if OK Returns -8039 if Invalid load definition.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.DeleteWindDefinition(1)
```

EndLoadMerging() [source]

End Load Merging

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.EndLoadMerging()
```

GetActiveLoad() [source]

Returns the current load case number.

Returns:

Returns active load case number ID. Else -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetActiveLoad()
```

GetAssignmentListForLoadType(*LoadType: int, LoadIndex: int*) [source]

Return the list of entities that have been assigned to a load command in the active load case (see function SetLoadActive). This command is identified as the LoadType, which has a defined reference number (see below) and an index number starting from 0. That is, if a load command has been defined 10 times in the load case, then index 0 identifies the first instance of the command and index 9 identifies the 10th.

Parameters:

- **loadType** (*int*) –

Value	LoadType	○ Value | LoadType		
4000	SelfWeight		3275	Uniform Force (Physical)
3110	Nodal Load (Node)		3280	Uniform Moment (Physical)
3120	Nodal Load (Inclined)		3285	Concentrated Force (Physical)
3910	Nodal Load (Support Displacement)		3290	Concentrated Moment (Physical)
3210	Uniform Force		3295	Trapezoidal (Physical)
3220	Uniform Moment		3310	Pressure on full plate
3230	Concentrated Force		3310	Concentrated Load (Plate)
3240	Concentrated Moment		3310	Partial plate pressure load
3250	Linear Varying		3320	Trapezoidal (Plate)
3260	Trapezoidal		3322	Solid
3260	Hydrostatic		3710	Temperature
3620	Pre/Post Stress		3720	Strain
3810	Fixed End		3721	Strain Rate
3530	FloorLoadGroup		3410	Area
3554	OneWayFloorLoadGroup			

- **loadIndex** (*int*) – Load item index of specified load type (Zero based). Program returns the first one start from loadIndex if exist the same load type in specified load case.

Returns:

Returns List of Entities number ID(s)

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetAssignmentListForLoadType(1, 0)
```

GetAttribute(*lLoadCase: int*) [source]

Gets load attribute information of specified load case.

Parameters:

lLoadCase (*int*) – Load case reference ID.

Returns:

Returns 0 if OK. Returns -1 if General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetAttribute(1)
```

GetBeamCountAtFloor(*varfMinX: float, varfMaxX: float, varfMinY: float, varfMaxY: float, varfMinZ: float, varfMaxZ: float, varnDirection: int*) [source]

Get the beam count at the specific floor.

Parameters:

- **varfMinX** (*float*) – varfMinX X range start (in global coordinate)
- **varfMaxX** (*float*) – varfMaxX X range end (in global coordinate)
- **varfMinY** (*float*) – varfMinY Y range start (in global coordinate)
- **varfMaxY** (*float*) – varfMaxY Y range end (in global coordinate)
- **varfMinZ** (*float*) – varfMinZ Z range start (in global coordinate)
- **varfMaxZ** (*float*) – varfMaxZ Z range end (in global coordinate)
- **varnDirection** (*int*) – varnDirection Direction(1 for XRange, 2 for YRange, 3 for ZRange).

Returns:

The beam count at the specific floor.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetBeamCountAtFloor(0, 10, 0, 10, 0, 10, 1)
```

GetConcForceCount(*nBeamNo: int*) [source]

Get number of concentrated force(s) present for the specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of concentrated force item(s) applied. Returns -1 if general error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetConcForceCount(1)
```

GetConcForces(*nBeamNo: int*) [source]

Returns the concentrated force(s) with all the parameters for the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Return a list of tuple in which tuple consist of load direction, Magnitude of the concentrate force, distance from the start of the member to concentrated force or moment, Perpendicular distance from the member shear center to the local plane of loading respectively. Load direction will be represented numerically - 1 to 6 for LocalX, LocalY, LocalZ, GlobalX, GlobalY and GlobalZ, respectively

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetConcForces(1)
```

GetConcMomentCount(*nBeamNo: int*) [source]

Gets number of concentrated moment(s) present for the specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of concentrated moment item(s) applied. Returns -1 if general error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetConcMomentCount(1)
```

GetConcMoments(*nBeamNo: int*) [source]

Returns the concentrated moments(s) with all the parameters for the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Return a list of tuple in which tuple consist of load direction, Magnitude of the concentrate moment, distance from the start of the member to concentrated force or moment, Perpendicular distance from the member shear center to the local plane of loading respectively. Load direction will be represented numerically - 1 to 6 for LocalX, LocalY, LocalZ, GlobalX, GlobalY and GlobalZ, respectively

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetConcMoments(1)
```

GetElementConcLoadCount(*varPlateNo: int*) [source]

Returns the number of concentrated load for specified plate.

Parameters:

varPlateNo (*int*) – Plate number ID.

Returns:

Returns the number of concentrated load on specified plate. Returns -1 if General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetElementConcLoadCount(1)
```

GetElementConcLoads(*varPlateNo: int*) [source]

Returns the concentrated load(s) with all the parameters for the specified plate.

Parameters:

varPlateNo (*int*) – Plate number ID.

Returns:

Returns a list of tuple in which tuple consist of load direction, pressure, the first d x coordinate, the first d y coordinate respectively. Load direction will be represented numerically - 1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY and ProjectedZ respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetElementConcLoads(1)
```

GetElementLoadInfo(*loadIndex: int*) [source]

Gets element load information generated by specified load item in specified load case. Please select the loadCase from UI (ElementLoadPressure only supported).

Parameters:

loadIndex (*int*) – Load item index (Zero based)

Returns:

Returns a list of tuple in which tuple consist of load direction, element pressures : dW1, dW2, dW3 and dW4, element force distances: dX1, dY1, dX2 and dY2 respectively. Load direction will be represented numerically - = 0 to 8 for LocalX, LocalY, LocalZ, GlobalX, GlobalY and GlobalZ, ProjectedX, ProjectedY, ProjectedZ respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetElementLoadInfo(1)
```

GetElementPressureLoadCount(*varPlateNo: int*) [source]

Gets the number pressure load(s) for the specified plate.

Parameters:

varPlateNo (*int*) – Plate number ID.

Returns:

Returns the number of pressure load(s). Returns -1 if General error.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetElementPressureLoadCount(1)
```

GetElementPressureLoads(*varPlateNo: int*) [source]

Returns the pressure load(s) with all the parameters for the specified plate.

Parameters:

varPlateNo (*int*) – Plate number ID.

Returns:

Returns a list of tuple in which tuple consist of load direction, Magnitude of the pressure load(s), Top-Left coordinate X (local), Top-Left coordinate Y (local), Bottom-Right coordinate X (local) and Bottom-Right coordinate Y (local) respectively. Load direction will be represented numerically - 1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY and ProjectedZ respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetElementPressureLoads(1)
```

GetEnvelopeCount() [source]

Returns number of Envelopes defined.

Returns:

Total Number of load Envelopes present.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetEnvelopeCount()
```

GetEnvelopeIDs() [source]

Gets the list of Loads Envelope IDs present in the staad file.

Returns:

Envelope ID(s)

Return type:

List of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetEnvelopeIDs()
```

GetInfluenceArea(*varfMinX: float, varfMaxX: float, varfMinY: float, varfMaxY: float, varfMinZ: float, varfMaxZ: float, varnDirection: int*) [source]

Returns a dictionary of beam to influence area at the specific floor.

Parameters:

- **varfMinX** (*float*) – X range start (in global coordinate).
- **varfMaxX** (*float*) – X range end (in global coordinate).
- **varfMinY** (*float*) – Y range start (in global coordinate).
- **varfMaxY** (*float*) – Y range end (in global coordinate).
- **varfMinZ** (*float*) – Z range start (in global coordinate).
- **varfMaxZ** (*float*) – Z range end (in global coordinate).
- **varnDirection** (*int*) – Direction(1 for XRange, 2 for YRange, 3 for ZRange).

Returns:

Returns dictionary have beam id to influence area data.

Return type:

Dictionary

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetInfluenceArea(0, 10, 0, 10, 0, 10, 1)
```

GetLinearVaryingLoadCount(*nBeamNo: int*) [source]

Returns number of linear varying load(s) present for the specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of linear varying load item(s) applied. Returns -1 if General Error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLinearVaryingLoadCount(1)
```

GetLinearVaryingLoads(*nBeamNo: int*) [source]

Returns parameters for defining linear varying loads for specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns a list of tuple in which tuple consist of load direction, load at the start of the member, load at the end of the member, Load in the middle of the member (for triangular load) respectively. Load direction will be represented numerically - 1 to 3 for local X, Y and Z, respectively

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLinearVaryingLoads(1)
```

GetListSizeForLoadType(*loadType: int, loadIndex: int*) [source]

Gets number of entities to vwhich specified Load Type and load index.

Parameters:

- **loadType** (*int*) –

Value	LoadType	○	Value | LoadType	
4000	SelfWeight		3275	Uniform Force (Physical)
3110	Nodal Load (Node)		3280	Uniform Moment (Physical)
3120	Nodal Load (Inclined)		3285	Concentrated Force (Physical)
3910	Nodal Load (Support Displacement)		3290	Concentrated Moment (Physical)
3210	Uniform Force		3295	Trapezoidal (Physical)
3220	Uniform Moment		3310	Pressure on full plate
3230	Concentrated Force		3310	Concentrated Load (Plate)
3240	Concentrated Moment		3310	Partial plate pressure load
3250	Linear Varying		3320	Trapezoidal (Plate)
3260	Trapezoidal		3322	Solid
3260	Hydrostatic		3710	Temperature
3620	Pre/Post Stress		3720	Strain
3810	Fixed End		3721	Strain Rate
3530	FloorLoadGroup		3410	Area
3554	OneWayFloorLoadGroup			

- **loadIndex** (*int*) – Load item index of specified load type (Zero based). Program returns the first one start from loadIndex if exist the same load type in specified load case.

Returns:

Returns the number of entities.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetListSizeForLoadType(1, 0)
```

`GetLoadAndFactorForCombination(varLoadCombNo: int)` [source]

Get number of concentrated force(s) present for the specified beam.

Parameters:

varLoadCombNo (*int*) – Combination Load case reference number ID.

Returns:

Returns a Tuple consisting of a list of load case reference number IDs and list of multiplication factors. For SRSS, in multiplication factor list, an extra element is added, at the end. This factor represents overall multiplication factor for SRSS combination.

Return type:

Tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadAndFactorForCombination(1)
```

GetLoadCaseTitle(*varLoadNo: int*) [source]

Returns title of the specified load case as a text string. Input 0 to retrieve title of current active load case or reference load case

Parameters:

varLoadNo (*int*) – The load case string title.

Returns:

Returns the load case string title. Returns “NONE” if load case varLoadNo not found.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadCaseTitle(1)
```

GetLoadCombinationCaseCount() [source]

Returns the total number of load combination cases in the current structure.

Returns:

Total number of load combination cases.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadCombinationCaseCount()
```

GetLoadCombinationCaseNumbers() [source]

Retrieves all load combination case numbers.

Returns:

List of load case reference number IDs.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadCombinationCaseNumbers()
```

GetLoadCountInLoadList(*varLoadListIndex: int*) [source]

Gets the number of load case(s) in specified load list.

Parameters:

varLoadListIndex (*int*) – Load list index.

Returns:

The number of Load Case(s) in specified Load List.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadCountInLoadList(1)
```

GetLoadEnvelopeDetails(*EnvNo: int*) [source]

Returns

Parameters:

EnvNo (*int*) – Load Envelope reference ID.

Returns:

Returns a tuple containing EnvelopeType and NumberofLoadCasesInEnvelope information respectively. Type of Load Envelope +——-+——————-+ | Value | Load Envelop Type | +=======+===================+ | 0 | NONE | +——-+——————-+ | 1 | STRESS | +——-+——————-+ | 2 | SERVICEABILITY | +——-+——————-+ | 3 | COLUMN | +——-+——————-+ | 4 | CONNECTION | +——-+——————-+ | 5 | STRENGTH | +——-+——————-+ | 6 | TEMPORARY | +——-+——————-+

Return type:

Tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetEnvelopeDetails(1)
```

GetLoadItemType(*loadCaseNo: int, loadItemIndex: int*) [source]

Returns the load item type for the specified loadIndex and loadCase.

Parameters:

- **loadCaseNo** (*int*) – Load case number.
- **loadItemIndex** (*int*) – Load item index (Zero based).

Returns:

Returns LoadItemType for the specified loadIndex and loadCase. Returns 0 if LoadCase/LoadItemIndex Not Found. Return Values and LoadItem Type +——-+————————————+—+——-+——————————+ | Value | LoadItem Type | * | Value | LoadItem Ty +======+=====================================+===+=======+=============================== | 4000 | SelfWeight | | 3520 | FloorLoadZrange | +——-+————————————+—+——-+——————————+ | 3110 | Nodal (Node) | | 3530 | FloorLoadGroup | +——-+————————————+—+——-+——————————+ | 3120 | Nodal Load (Incline 3551 | OneWayFloorLoadXrange | +——-+————————————+—+——-+——————————+ | 3910 | Nodal Load (Suppor Displacement) | | 3552 | OneWayFloorLoadYrange | +——-+————————————+—+——-+——————————+ | 3312 | No Load (Region node load) | | 3553 | OneWayFloorLoadZrange | +——-+————————————+—+——-+——————————+ | 3210 | Uniform Force | | 3554 | OneWayFloorLoadGroup | +——-+————————————+—+——-+——————————+ | 322 Uniform Moment | | 3310 | Pressure on full plate | +——-+————————————+—+——-+——————————+ | 3230 |

Concentrated Force | | 3311 | Concentrated Load (Plate) | +——-+————————————+—+——-+————————————+ | 324
Concentrated Moment | | 3312 | Partial plate pressure load | +——-+————————————+—+——-+————————————+ | 3
| Linear Varying | | 3320 | Trapezoidal (Plate) | +——-+————————————+—+——-+————————————+ | 3260 | Trapezoid
3322 | Solid | +——-+————————————+—+——-+————————————+ | 3261 | Hydrostatic | | 3710 | Temperature | +——
————————————+—+——-+————————————+ | 3620 | Pre/Post Stress | | 3720 | Strain | +——-+————————————
+—+——-+————————————+ | 3810 | Fixed End | | 3721 | Strain Rate | +——-+————————————+—+——-+
————————————+ | 3275 | Uniform Force (Physical) | | 4400 | UBC Load | +——-+————————————+—+——-+
————————————+ | 3280 | Uniform Moment (Physical) | | 4600 | Wind Load | +——-+————————————+—+——-+
————————————+ | 3285 | Concentrated Force (Physical) | | 4610 | Wind Load Dynamic | +——-+————————————+—+
——-+————————————+ | 3290 | Concentrated Moment (Physical) | | 4405 | IbcLoad | +——-+————————————+—+—
+————————————+ | 3295 | Trapezoidal (Physical) | | 4410 | 1893Load | +——-+————————————+—+——-+
————————————+ | 3410 | Area | | 4500 | AijLoad | +——-+————————————+—+——-+————————————+ | 351
FloorLoadYrange | | 4510 | ColombianLoad | +——-+————————————+—+——-+————————————+ | 3511 |
FloorLoadXrange | | 4520 | CFELoad | +——-+————————————+—+——-+————————————+ | 4570 | TurkishLoad | | 45
RPALoad | +——-+————————————+—+——-+————————————+ | 4575 | GB50011Load | | 4540 | NTCLoad | +——-+
————————————+—+——-+————————————+ | 4576 | Colombian2010Load | | 4550 | NRCLoad | +——-+
————————————+—+——-+————————————+ | 4820 | TimeHistoryLoad | | 4560 | NRCLoad2005 | +——-+
————————————+—+——-+————————————+ | 4651 | Snow Load Data | | 4561 | NRCLoad2010 | +——-+
————————————+—+——-+————————————+ | 4201 | Repeat load data | | 4100 | Spectrum Load | +——-+
————————————+—+——-+————————————+ | 4223 | Notional Load Data | | 4700 | Calulate Natural Frequency | +——
————————————+—+——-+————————————+ | 4220 | Reference Load | | 4710 | Modal Calculation Requested | +——-
————————————+—+——-+————————————+ | 4101 | Spectrum Data | | 4222 | Notional Load | +——-+
————————————+—+——-+————————————+ | 4701 | Calulate Rayleigh Frequency | | 4650 | Snow Load | +——-+
————————————+—+——-+————————————+ | 4200 | Repeat load | | | | +——-+————————————+—+——-
————————————+

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadItemType(1, 1)
```

GetLoadItemsCount(*LoadCaseNo: int*) [source]

Returns the number of loaditems in the specified load case.

Parameters:

loadCaseNo (*int*) – Load case number.

Returns:

Returns the number of loaditems in the specified load case. Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadItemsCount(1)
```

GetLoadListCount() [source]

Gets the number of existing load list(s)

Returns:

Returns the number of load list(s). Returns -1 if General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadListCount()
```

`GetLoadListfromLoadEnvelope(EnvNo: int)` [source]

Gets the list of primary load case reference Ids present in the load envelope passed.

Parameters:

EnvNo (*int*) – Load Envelope reference ID

Returns:

(Primary) load case(s) reference ID(s).

Return type:

List of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadListfromLoadEnvelope(1)
```

`GetLoadType(varLoadNo: int)` [source]

Returns primary load case category(s) as an long value.

Parameters:

varLoadNo (*int*) – Primary load case reference ID. Pass in 0 to get information about current active load case or reference load case

Returns:

Returns 0 if Dead. Returns 1 if Live. Returns 2 if Roof Live. Returns 3 if Wind. Returns 4 if Seismic-H. Returns 5 if Seismic-V. Returns 6 if Snow. Returns 7 if Fluids. Returns 8 if Soil. Returns 9 if Rain. Returns 10 if Ponding. Returns 11 if Dust. Returns 12 if Traffic. Returns 13 if Temperature. Returns 14 if Imperfection. Returns 15 if Accidental. Returns 16 if Flood. Returns 17 if Ice. Returns 18 if Wind Ice. Returns 19 if Crane Hook. Returns 20 if Mass. Returns 21 if Gravity. Returns 22 if Push. Returns 23 if None. Returns -1 if General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadType(1)
```

`GetLoadTypeCount(LoadType: int)` [source]

Gets the number of load(s) with specified Load Type in active Load Case.

Parameters:

loadType (*int*) –

Value	LoadType	•	Value | LoadType	
4000	SelfWeight		3275	Uniform Force (Physical)
3110	Nodal Load (Node)		3280	Uniform Moment (Physical)
3120	Nodal Load (Inclined)		3285	Concentrated Force (Physical)
3910	Nodal Load (Support Displacement)		3290	Concentrated Moment (Physical)
3210	Uniform Force		3295	Trapezoidal (Physical)
3220	Uniform Moment		3310	Pressure on full plate
3230	Concentrated Force		3310	Concentrated Load (Plate)
3240	Concentrated Moment		3310	Partial plate pressure load
3250	Linear Varying		3320	Trapezoidal (Plate)
3260	Trapezoidal		3322	Solid
3260	Hydrostatic		3710	Temperature
3620	Pre/Post Stress		3720	Strain
3810	Fixed End		3721	Strain Rate
3530	FloorLoadGroup		3410	Area
3554	OneWayFloorLoadGroup			

Returns:

Returns the number of load(s). Returns 0 if loadCaseNo not found.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadTypeCount(1)
```

GetLoadsInLoadList(*varLoadListIndex: int*) [source]

Gets the load case(s) in specified load list.

Parameters:

varLoadListIndex (*int*) – Load list index(Starts from one).

Returns:

Load Case reference IDs list.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetLoadsInLoadList(1)
```

GetMemberLoadInfo(*LoadIndex: int*) [source]

Gets member load(s) information generated by specified load item in specified load case.

Parameters:

loadIndex (*int*) – Load item index (Zero based)

Returns:

- direction: int (Load direction will be represented numerically - 1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ, respectively.)
- member force parameters: List [dW1, dW2, dW3]
- member force distances: List [dD1, dD2, dD3]

Return type:

tuple containing

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetMemberLoadInfo(1)
```

GetNoLoadFactorDirectionInNotionalLoad(*nIndex: int*) [source]

Gets the no of factor for specified Notional load.

Parameters:

nIndex (*int*) – The index for Notional load.

Returns:

Returns the factor for specified Notional load. Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNoLoadFactorDirectionInNotionalLoad(1)
```

GetNoLoadFactorInRepeatLoad(*nIndex: int*) [source]

Returns the number of load and factor pairs associated with a given repeat load command in the active load case.

Parameters:

nIndex (*int*) – The index(One based) for repeat load.

Returns:

Returns number of load and factor pairs associated with a given repeat load command in the active load case. Returns -1 if case of invalid repeat load index.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNoLoadFactorInRepeatLoad(1)
```

GetNoOfLoadAndFactorPairsForCombination(*varLoadCombNo: int*) [source]

Gets the number of load case(s) applied with multiplication factor in specified load combination.

Parameters:

varLoadCombNo (*int*) – Combination Load case reference number ID.

Returns:

Returns the number of load cases in specified load combination.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNoOfLoadAndFactorPairsForCombination(1)
```

GetNoOfSetsInReferenceLoad(*nIndex: int*) [source]

Returns the number of reference load case-factor sets in a specified reference load item.

Parameters:

nIndex (*int*) – Index of the reference load case item.

Returns:

Number of sets in the reference load item. Returns -1 in case of an error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNoOfSetsInReferenceLoad(1)
```

GetNodalLoadCount(*nNodeNo: int*) [source]

Returns number of nodal loads present for the specified node.

Parameters:

nNodeNo (*int*) – Node Id

Returns:

Returns the number of node(s). Else -1 if general error (perhaps load case not found).

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNodalLoadCount(1)
```

GetNodalLoadInfo(*LoadIndex: int*) [source]

Gets nodal load(s) generated by specified load item in specified load case.

Parameters:

loadIndex (*int*) – Load item index.

Returns:

Returns a list of 5 nodal forces - FX, FY, FZ, MX, MY and MZ which are placed respectively.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNodalLoadInfo(1)
```

GetNodalLoads(*nNodeNo: int*) [source]

Returns tuple of list of forces in X direction, forces in Y direction, forces in Z direction, moments in X direction, moments in Y direction and moments in Z direction respectively.

Parameters:

nNodeNo (*int*) – Node Id

Returns:

Returns a tuple of list of forces in X direction, forces in Y direction, forces in Z direction, moments in X direction, moments in Y direction and moments in Z direction respectively.

Return type:

Tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNodalLoads(1)
```

GetNotionalLoadByIndex(*nIndex: int*) [source]

Gets load case(s), direction(s) and factor(s) for specified Notional load.

Parameters:

nIndex (*int*) – The index for Notional load.

Returns:

Returns a list of tuple in which tuple consist of load direction, load case reference ID(s) in VARIANT array - (if +ve values = Primary Load Cases or -ve values = Reference Load Cases) , load factors respectively. Load direction will be represented numerically - = 1 to 3 for X, Y and Z direction respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNotionalLoadByIndex(1)
```

`GetNotionalLoadCount()`

[source]

Returns the number of Notional load.

Returns:

Returns the number of Notional load Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetNotionalLoadCount()
```

`GetPrimaryLoadCaseCount()`

[source]

Returns the total number of primary load cases in the current structure.

Returns:

Total number of primary load cases.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetPrimaryLoadCaseCount()
```

`GetPrimaryLoadCaseNumbers()`

[source]

Retrieves all primary load case numbers.

Returns:

List of load case reference number IDs.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetPrimaryLoadCaseNumbers()
```

`GetReferenceLoadByIndex(`*`nIndex: int`*`)`

[source]

Retrieves a dictionary of load case numbers and their corresponding factors for a given reference load case.

Parameters:

nIndex (*int*) – Index of the reference load.

Returns:

tuple of load case number factor lists. [[loadcase1, loadcase2, …], [factor1, factor2, …]]

Return type:

tuple of lists

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadByIndex(1)
```

`GetReferenceLoadCaseCount()` [source]

Returns the number of reference load case items in the currently active load case.

Returns:

Number of reference load case items. Returns -1 in case of an error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadCaseCount()
```

`GetReferenceLoadCaseNumbers()` [source]

Retrieves reference load case number IDs from Reference Load Definitions.

Returns:

List of reference load case IDs.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadCaseNumbers()
```

`GetReferenceLoadCaseTitle(varLoadNo: int)` [source]

Returns the title of a reference load case.

Parameters:

varLoadNo (*int*) – Reference load number.

Returns:

Title of the reference load case.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadCaseTitle(1)
```

`GetReferenceLoadCount()` [source]

Returns the number of reference load items in the currently active load case.

Returns:

Number of reference load items. Returns -1 in case of an error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadCount()
```

`GetReferenceLoadType(varLoadNo: int)` [source]

Returns the type of a reference load.

Parameters:

varLoadNo (*int*) – Reference load number.

Returns:

Reference load type (0 to 23). Returns -1 in case of an error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetReferenceLoadType(1)
```

`GetRepeatLoadByIndex(nIndex: int)` [source]

Returns the dictionary of load case IDs to load factors for a given repeat load command in the active load case.

Parameters:

nIndex (*int*) – The index(One based) for repeat load.

Returns:

Returns a dictionary of load case ID to load factor.

Return type:

dictionary

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetRepeatLoadByIndex(1)
```

GetRepeatLoadCount() [source]

Returns the number of repeat load commands in the active load case.

Returns:

Returns the number of repeat load commands in the active load case. Returns 0 if General Error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetRepeatLoadCount()
```

GetTrapLoadCount(*nBeamNo: int*) [source]

Returns number of trapezoidal load(s) present for the specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of trapezoidal load item(s) applied. Returns -1 if general error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetTrapLoadCount(1)
```

GetTrapLoads(*nBeamNo: int*) [source]

Returns the trapezoidal load(s) with all the parameters for the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Return a list of tuple in which tuple consist of load direction, Load at the start of the member, Load at the end of the member, distance from the start of the member to loading starting point, distance from the end of the member to loading stopping point respectively. Load direction will be represented numerically - 1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ, respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetTrapLoads(1)
```

GetUDLLoadCount(*nBeamNo: int*) [source]

Returns the number of uniformly distributed load(s) present for the specified beam.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of uniformly distributed load item(s) applied. Returns -1 if general error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetUDLLoadCount(1)
```

`GetUDLLoads(nBeamNo: int)` [source]

Gets the uniformly distributed load(s) with all the parameters for the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Return a tuple of lists in which each list consist of - load directions (1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ, respectively.) - magnitude of uniform force - distance from start of member to the start of load - distance from start of member to the end of load - perpendicular distance from the member shear center to the local plane of loading respectively. [[dirL1, dirL2,..], [forceL1, forceL2,..], [dst_startL1, dst_startL2,..], [dst_endL1, dst_endL2,..], [dst_perpendicularL1, dst_perpendicularL2,..]]

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetUDLLoads(1)
```

`GetUNIMomentCount(nBeamNo: int)` [source]

Returns the count of uniformly distributed (UNI) moment applied to the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Returns the number of uniformly distributed (UNI) moment item(s) applied. Returns -1 if general error

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetUNIMomentCount(1)
```

`GetUNIMoments(nBeamNo: int)` [source]

Returns the uniformly distributed (UNI) moments with all the parameters for the specified member.

Parameters:

nBeamNo (*int*) – Beam number ID.

Returns:

Return a list of tuple in which tuple consist of load direction, magnitude of uniform moment, distance from start of member to the start of load, distance from start of member to the end of load, perpendicular distance from the member shear center to the local plane of loading respectively. Load direction will be represented numerically - 1 to 9 for LocalX, LocalY, LocalZ, GlobalX, GlobalY, GlobalZ, ProjectedX, ProjectedY, ProjectedZ, respectively.

Return type:

List of tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.GetUNIMoments(1)
```

IsCombinationCase(*nLoadCase: int*) [source]

Checks if specified load case is combination load case.

Parameters:

nLoadCase (*int*) – Load case reference ID

Returns:

Returns 1 if YES Returns 0 if NO Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.IsCombinationCase(1)
```

IsDynamicLoadIncluded(*nLoadCase: int*) [source]

Checks if dynamic load included in specified load case.

Parameters:

nLoadCase (*int*) – Load case reference ID

Returns:

Returns 1 if YES Returns 0 if NO Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.IsDynamicLoadIncluded(1)
```

MergeLoadsOnBeam(*varBeamToKeep: int, varBeamToMerge: int*) [source]

Merge Load from beam to merge.

Parameters:

- **varBeamToKeep** (*int*) – Beam Id where load to not merge.
- **varBeamToMerge** (*int*) – Beam Id to where load to merge.

Returns:

Returns 1 (TRUE) if Successful Returns 0 (FALSE) if General Error.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.MergeLoadsOnBeam(1, 2)
```

`ModifySeismicDefinitionParams(varParamName: str, varValue: float)` [source]

Modifies or adds a seismic parameter in the existing seismic definition.

Parameters:

- **varParamName** (*string*) – Parameter name for the corresponding code in the seismic definition.
- **varValue** (*float*) –

  Value corresponding to the above parameter:

Seismic Code	Parameters
ALGERIAN: RPA	A Q RX RZ STYPE CT CRDAMP PX PZ
CANADIAN: NRC-1995	V ZA ZV RX RZ I F CT PX PZ
CANADIAN: NRC-2005	SA1 SA2 SA3 SA4 IE SCLASS MVX MVZ JX JZ RDX RDZ ROX ROZ CT PX PZ FA FV
CANADIAN: NRC-2010	SA1 SA2 SA3 SA4 I SCLASS MVX MVZ RDX RDZ ROX ROZ CTX CTZ PX PZ FA FV STX STZ MD
CHINESE: GB50011-2001	INTENSITY FREQUENT RARE GROUP SCLASS DAMP DELN SF PX PZ GFACTOR Note: For CHINESE: GB50011-2001 FREQUENT/RARE parameter value will be 0, 1 respectively
CHINESE: GB50011-2010	INTENSITY FREQUENT FORTIFIED RARE GROUP SCLASS DAMP GFACTOR DELN SF PX PZ Note: For CHINESE: GB50011-2010 FREQUENT/FORTIFIED/RARE parameter value will be 0, 1 and 2 respectively
COLOMBIAN: NSR 98	ZONE I S
COLOMBIAN: NSR 2010	AA AV FA FV I CT PX PZ ALPHA
IBC 2000	SDS SD1 S1 I RX RZ SCLASS CT PX PZ
IBC 2003	SDS SD1 S1 I RX RZ SCLASS CT PX PZ
IBC 2006	SS S1 ZIP I RX RZ SCLASS CTX CTZ PX PZ LAT LONG TL FA FV XX XZ
IBC 2012	SS S1 ZIP I RX RZ SCLASS CTX CTZ PX PZ LAT LONG TL FA FV XX XZ
IBC 2015	SS S1 ZIP I RX RZ SCLASS CTX CTZ PX PZ LAT LONG TL FA FV XX XZ
IBC 2018	SS S1 ZIP I RX RZ SCLASS CTX CTZ PX PZ LAT LONG TL FA FV XX XZ Note: For IBC 2006 - 2018 Please provide any one of ZIP OR LAT LONG OR SS S1
INDIAN: IS 1893-1984	ZONE K I B PX PZ
INDIAN: IS 1893-2002/2005	ZONE RF I SS ST DM PX PZ DT GL SA DF CS AX ES CV DV
INDIAN: IS 1893-2016	ZONE RF I SS ST DM PX PZ DT GL SA DF HT DX DZ
INDIAN: IS 1893(Part4) 2015	ZONE RF I SS ST DM PX PZ SA DF
JAPANESE (AIJ)	ZONE CO TC ALPHA
MEX: CFE-1993	ZONE QX QZ GROUP STYPE REGULAR TS PX PZ
MEX: NTC-1987	ZONE QX QZ GROUP SHADOWED REGULAR REDUCE PX PZ Note: For SHADOWED, REGULAR and REDUCE parameter value will be 0 or 1 respectively
TURKISH	A TA TB I RX RZ CT PX PZ
UBC 1985	ZONE I K TS

Seismic Code	Parameters
UBC 1994	ZONE I RWX RWZ S CT PX PZ
UBC 1997	ZONE I RWX RWZ STYPE CT PX PZ NA NV

Returns:

Returns 0 if OK Returns -1 if general error.

Return type:

int

Example

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.ModifySeismicDefinitionParams('ZONE', 0.2)
```

RemoveAttribute(*lLoadCase: int*) [source]

Removes the load attribute specified by lLoadCase.

Parameters:

lLoadCase (*int*) – Load case reference ID

Returns:

Returns 0 if OK Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.RemoveAttribute(1)
```

RemoveLoadCasesFromEnvelop(*varEnvNo: int*, *varLoadCaseList: list*) [source]

Removes a list of primary load case(s) from an existed load envelop.

Parameters:

- **varEnvNo** (*int*) – Load Envelop reference ID
- **varLoadCaseList** (*list of int*) – Load cases reference IDs list.

Returns:

Returns 0 if OK Returns -1 if general error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.RemoveLoadCasesFromEnvelop(1, [2, 3])
```

SetASDLoadAttribute(*loadCaseRefID: int, strengthType: int, allowStressIncrease: bool*) [source]

Sets Allowable Stress Design (ASD) load attribute.

Parameters:

- **loadCaseRefID** (*int*) – Load case reference ID.
- **strengthType** (*int*) –

  Strength Type :

Value	Integer
STRENGTH_TYPE_NONE	0
NORMAL_ASD_WORKING_STRESS_FORCES_WITHOUT_P_DELTA	1
NORMAL_ASD_WORKING_STRESS_FORCES_WITH_P_DELTA	2
STRENGTH_TYPE_OF_FORCES_WITHOUT_P_DELTA	3
STRENGTH_TYPE_OF_FORCES_WITH_P_DELTA	4
COLUMN_ONLY_STRENGTH_TYPE_OF_FORCES_WITHOUT_P_DELTA	5
COLUMN_ONLY_STRENGTH_TYPE_OF_FORCES_WITH_P_DELTA	6

- **allowStressIncrease** (*bool*) – Allow 1/3 stress increase in ASD.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.SetASDLoadAttribute(1, 1, True)
>>> print(result)
```

SetLSDLoadAttribute(*loadCaseRefID: int*) [source]

Sets Limit State Design (LSD) load attribute.

Parameters:

loadCaseRefID (*int*) – Load case reference ID.

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.SetLSDLoadAttribute(1)
>>> print(result)
```

SetLoadActive(*loadNumber: int*) [source]

Activates the specified load number to allow adding or removing load items.

Parameters:

loadNumber (*int*) – Load case reference number ID.

Returns:

True if the load case was successfully activated. False if an error occurred.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.SetLoadActive(1)
>>> print(result)
```

SetLoadType(*LoadCaseNumber: int*, *LoadType: int*) [source]

Set load type to load case for considering load combination.

Parameters:

- **loadCaseNumber** (*int*) – The load case reference number ID.
- **loadType** (*int*) –

  Type of the load.:

Value	Load Type	Value	Load Type
0	Dead	12	Traffic
1	Live	13	Temp
2	Roof Live	14	Imperfection
3	Wind	15	Accidental
4	Seismic-H	16	Flood
5	Seismic-V	17	Ice
6	Snow	18	Wind Ice
7	Fluids	19	Crane Hook
8	Soil	20	Mass
9	Rain	21	Gravity
10	Ponding	22	Push
11	Dust	23	None

Returns:

Returns 0 OK. Returns -1 General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.SetLoadType(1, 0)
>>> print(result)
```

SetReferenceLoadActive(*nLoadCaseNo: int*) [source]

Activates a reference load case to allow operations on its items.

Parameters:

nLoadCaseNo (*int*) – Reference load case ID in Load Case Details.

Returns:

Reference load case number ID. Returns -1 in case of an error. Returns -8002 if the load case is not found.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Load.SetReferenceLoadActive(2)
>>> print(result)
```

SplitLoadsOnBeam(*varBeamOld: int, varBeamNew: int*) [source]

Split Load from BeamOld to BeamNew.

Parameters:

- **varBeamOld** (*int*) – Old Beam Id
- **varBeamNew** (*int*) – New Beam Id

Returns:

Returns 1 (TRUE) if Successful Returns 0 (FALSE) if General Error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Load.SplitLoadsOnBeam(1, 2)
```

__init__(*staadObj*) [source]