Geometry

Contents

class `openstaadpy.os_analytical.osgeometry.OSGeometry` [source]

Bases: `object`

`AddBeam(nNodeStart: int, nNodeEnd: int)` [source]

Add a beam/member with specified nodes and return the assigned beam number.

Parameters:

- **nNodeStart** (*int*) – ID of the starting node.
- **nNodeEnd** (*int*) – ID of the ending node.

Returns:

Beam number assigned.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> beam_no = staad_obj.Geometry.AddBeam(1, 2)
>>> print(beam_no)
```

`AddCircularRegionToSurface(surfaceNo: int, x: float, y: float, z: float, radius: float, divisions: int, density: int, is_opening: bool = False)` [source]

Add a circular region or opening to a surface.

Parameters:

- **surfaceNo** (*int*) – Surface ID of the parametric surface to which the circular region will be added.
- **x** (*float*) – Global X coordinate of the center of the circular region.
- **y** (*float*) – Global Y coordinate of the center of the circular region.
- **z** (*float*) – Global Z coordinate of the center of the circular region.
- **radius** (*float*) – Radius of the circular region.
- **divisions** (*int*) – Number of divisions along the circular region.
- **density** (*int*) – Density of the circular region.
- **is_opening** (*bool*) – Whether the circular region is an opening or not.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.AddCircularRegionToSurface(1, 5.0, 5.0,
>>> print(result)
```

AddDensityLineToSurface(*surfaceNo: int, x1: float, y1: float, z1: float, density1: int, x2: float, y2: float, z2: float, density2: int, divisions: int*) [source]

Add a density line to a surface.

Parameters:

- **surfaceNo** (*int*) – Surface ID of the parametric surface to which the density line will be added.
- **x1** (*float*) – Global X coordinate of the start point of the density line.
- **y1** (*float*) – Global Y coordinate of the start point of the density line.
- **z1** (*float*) – Global Z coordinate of the start point of the density line.

- **density1** (*int*) – Density at the start point of the density line.
- **x2** (*float*) – Global X coordinate of the end point of the density line.
- **y2** (*float*) – Global Y coordinate of the end point of the density line.
- **z2** (*float*) – Global Z coordinate of the end point of the density line.
- **density2** (*int*) – Density at the end point of the density line.
- **divisions** (*int*) – Number of divisions along the density line.

Returns:

index (0 based) of the density line added.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.AddDensityLineToSurface(1, 0.0, 0.0, 0.
>>> print(result)
```

AddDensityPointToSurface(*surfaceNo: int, pointData*) [source]

Add a density point to a surface.

Parameters:

- **surfaceNo** (*int*)
- **pointData** (*object*)

Return type:

None

AddMultipleBeams(*incidences*) [source]

Add multiple beams at once.

Parameters:

incidences (*list of lists containing int*) – List of lists containing start and end node numbers for each beam. [[start1, end1], [start2, end2], ...]

Returns:

List of beam numbers assigned to the added beams.

Return type:

List

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> beam_ids = staad_obj.Geometry.AddMultipleBeams([[1,2],[2,3]])
>>> print(beam_ids)
```

`AddMultipleNodes(`*`coordinates`*`)` [source]

Add multiple nodes at once.

Parameters:

coordinates (*list of lists containing float or int*) – List of lists containing x, y, z coordinates for each node. [[x1, y1, z1], [x2, y2, z2], ...]

Returns:

List

Return type:

List of node numbers assigned to the added nodes.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> node_ids = staad_obj.Geometry.AddMultipleNodes([[0.0,0.0,0.0],[1.0,
>>> print(node_ids)
```

`AddMultiplePlates(`*`incidences`*`)` [source]

Add multiple plates at once.

Parameters:

incidences (*list*) – List of lists containing nodeA, nodeB, nodeC, nodeD for each plate. [[nodeA1, nodeB1, nodeC1, nodeD1], [nodeA2, nodeB2, nodeC2, nodeD2], …]

Returns:

List

Return type:

List of plate numbers assigned to the added plates.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> plate_ids = staad_obj.Geometry.AddMultiplePlates([[1,2,3,4],[2,3,4,
>>> print(plate_ids)
```

AddMultipleSolids(*incidences*) [source]

Add multiple solids at once.

Parameters:

incidences (*list*) – List of lists containing nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, nodeG, nodeH for each solid. [[nodeA1, nodeB1, nodeC1, nodeD1, nodeE1, nodeF1, nodeG1, nodeH1], [nodeA2, nodeB2, nodeC2, nodeD2, nodeE2, nodeF2, nodeG2, nodeH2], …]

Returns:

List

Return type:

List of solid numbers assigned to the added solids.

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> solid_ids = staad_obj.Geometry.AddMultipleSolids([[1,2,3,4,5,6,7,8]
>>> print(solid_ids)
```

AddNode(*x: float, y: float, z: float*) [source]

Add a node with specified coordinates and return the assigned node number.

Parameters:

- **x** (*float*) – X coordinate.
- **y** (*float*) – Y coordinate.
- **z** (*float*) – Z coordinate.

Returns:

Node number assigned.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> node_no = staad_obj.Geometry.AddNode(0.0, 0.0, 0.0)
>>> print(node_no)
```

AddParametricSurfaceToModel(*surfaceNo: int*) [source]

Add definition of the specified parametric surface to the model.

Parameters:

surfaceNo (*int*)

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> surface_id = staad_obj.Geometry.AddParametricSurfaceToModel(1)
>>> print(surface_id)
```

AddPlate(*nNodeA: int, nNodeB: int, nNodeC: int, nNodeD: int = 0*) [source]

Add a plate with specified nodes and return the assigned plate number.

Parameters:

- **nNodeA** (*int*) – Node A for plate connectivity.
- **nNodeB** (*int*) – Node B for plate connectivity.
- **nNodeC** (*int*) – Node C for plate connectivity.
- **nNodeD** (*int*) – Node D for plate connectivity.

Returns:

Plate number assigned.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> plate_no = staad_obj.Geometry.AddPlate(1, 2, 3, 4)
>>> print(plate_no)
```

AddPolygonalRegionToSurface(*surfaceNo: int, regionData*) [source]

Add a polygonal region to a surface.

Parameters:

- **surfaceNo** (*int*)
- **regionData** (*object*)

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.AddPolygonalRegionToSurface(1, regionData)
```

AddSolid(*nodeA: int, nodeB: int, nodeC: int, nodeD: int, nodeE: int, nodeF: int, nodeG: int = 0, nodeH: int = 0*) [source]

Add a solid element.

Parameters:

- **nodeA** (*int*) – ID of node A.
- **nodeB** (*int*) – ID of node B.
- **nodeC** (*int*) – ID of node C.
- **nodeD** (*int*) – ID of node D.
- **nodeE** (*int*) – ID of node E.
- **nodeF** (*int*) – ID of node F.
- **nodeG** (*int*) – ID of node G.
- **nodeH** (*int*) – ID of node H.

Returns:

ID number of the added solid.

Return type:

Int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> solidID = staad_obj.Geometry.AddSolid(1, 2, 3, 4, 5, 6, 7, 8)
>>> print(solidID)
```

BreakBeamsAtSpecificNodes(*nodeList: list*) [source]

Breaks beams that passes through the specified list of nodes and assigns same attributes to the newly added beams.

Parameters:

nodeList (*list of int*) – List of node IDs where beams should be broken.

Returns:

1. List of int : IDs of the broken beams.
2. List of int : IDs of the newly created beams.

Return type:

tuple of 2 lists

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> new_beams = staad_obj.Geometry.BreakBeamsAtSpecificNodes([1,2,3])
>>> print(new_beams)
```

ClearMemberSelection() [source]

Clear the current member selection.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.ClearMemberSelection()
```

ClearNodeSelection() [source]

Clear the current node selection.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.ClearNodeSelection()
```

ClearPhysicalMemberSelection() [source]

Clears the current selection of physical members.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.ClearPhysicalMemberSelection()
```

ClearPlateSelection() [source]

Clear the current plate selection.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.ClearPlateSelection()
```

ClearSolidSelection() [source]

Clear the current solid selection.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.ClearSolidSelection()
```

`CommitParametricSurfaceMesh(surfaceNo: int)` [source]

Merges the specified parametric mesh with the model

Parameters:

surfaceNo (*int*) – surface ID of the parametric surface to be merged with the model.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.CommitParametricSurfaceMesh(1)
>>> print(result)
```

`CreateBeam(nBeamNo: int, nNodeStart: int, nNodeEnd: int)` [source]

Create a beam/member with specified nodes.

Parameters:

- **nBeamNo** (*int*) – Member number ID to assign.
- **nNodeStart** (*int*) – ID of the starting node.
- **nNodeEnd** (*int*) – ID of the ending node.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateBeam(5, 1, 2)
```

CreateGroup(*group_type: int, group_name: str*) [source]

Create a new group with the specified name and type.

Parameters:

- **group_type** (*int*) –

  Type of the group:

Index	Group Type
1	Nodes
2	Members
3	Plates
4	Solids
5	Geometry (Members, Plates and Solids)
6	Floor (Floor beam)

- **group_name** (*str*) – Name of the group to create.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateGroup(1, "MyGroup")
```

CreateGroupEx(*groupType: int, groupName: str, entityList: list*)

Create a group with extended options. [source]

Parameters:

- **groupType** (*int*) –

  The int representating the corresponding group type as show in below table:

Index	Group Type
1	Nodes
2	Members
3	Plates
4	Solids
5	Geometry (Members, Plates and Solids)
6	Floor (Floor beam)

- **groupName** (*str*) – Name of the group.
- **entityList** (*list of int*) – List of entity IDs to include in the group.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateGroupEx(1, "GroupA", [1,2,3])
```

CreateMultipleBeams(*beam_ids: list, beam_incidences: list*)

Create multiple beams. [source]

Parameters:

- **beam_ids** (*list of int*) – Beam IDs for each beam. [BeamID1, BeamID2, BeamID3, ...]
- **beam_incidences** (*list of lists*) – List of [start_node, end_node] for each beam. [[start1, end1], [start2, end2], ...]

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateMultipleBeams(beam_ids, beam_incidences)
```

CreateMultipleNodes(*node_ids: list, nodeCoordinates: list*) [source]

Create multiple nodes.

Parameters:

- **node_ids** (*list of int*) – Node IDs for each node. [NodeID1, NodeID2, NodeID3, ...]
- **nodeCoordinates** (*list of lists*) – List of [x, y, z] coordinates for each node. [[x1, y1, z1], [x2, y2, z2], ...]

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateMultipleNodes(node_ids,nodeCoordinates)
```

CreateMultiplePlates(*plate_ids: list | int, plate_incidences: list*) [source]

Create multiple plates.

Parameters:

- **plate_ids** (*list of int or int*) – plate IDs for each plate. [PlateID1, PlateID2, PlateID3, ...]
- **plate_incidences** (*list of lists*) – List of lists containing incidences for each plate. [[NodeA1, NodeB1, NodeC1, NodeD1], [NodeA2, NodeB2, NodeC2, NodeD2], ...]

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateMultiplePlates(plateIds, plateIncidences)
```

CreateNode(*nNodeNo: int, x: float, y: float, z: float*) [source]

Create a node with specified coordinates and node number.

Parameters:

- **nNodeNo** (*int*) – Node number ID to assign.
- **x** (*float*) – X coordinate.
- **y** (*float*) – Y coordinate.
- **z** (*float*) – Z coordinate.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateNode(10, 0.0, 0.0, 0.0)
```

CreatePhysicalMember(*memberList: list*) [source]

Create a physical member from the currently selected members.

Parameters:

- **memberList** (*list of int*) – List of member IDs to include in the physical member.
- **physicalMemberName** (*str*) – Name of the physical member to create.

Returns:

Id of the created physical member.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.CreatePhysicalMember([1,2,3], "MyPhysic
>>> print(result)
```

`CreatePlate(nPlateNo: int, nNodeA: int, nNodeB: int, nNodeC: int, nNodeD: int = 0)` [source]

Create a plate with specified nodes.

Parameters:

- **nPlateNo** (*int*) – Plate number ID to assign.
- **nNodeA** (*int*) – Node A for plate connectivity.
- **nNodeB** (*int*) – Node B for plate connectivity.
- **nNodeC** (*int*) – Node C for plate connectivity.
- **nNodeD** (*int*) – Node D for plate connectivity.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreatePlate(1, 1, 2, 3, 4)
```

CreateSolid(*solidNo: int, nodeA: int, nodeB: int, nodeC: int, nodeD: int, nodeE: int, nodeF: int, nodeG: int = 0, nodeH: int = 0*) [source]

Create a solid element.

Parameters:

- **solidNo** (*int*) – Solid number ID to assign.
- **nodeA** (*int*) – ID of node A.
- **nodeB** (*int*) – ID of node B.
- **nodeC** (*int*) – ID of node C.
- **nodeD** (*int*) – ID of node D.
- **nodeE** (*int*) – ID of node E.
- **nodeF** (*int*) – ID of node F.
- **nodeG** (*int*) – ID of node G.
- **nodeH** (*int*) – ID of node H.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.CreateSolid(4, 1, 2, 3, 4, 5, 6, 7, 8)
```

DefineParametricSurface(*name: str, type: int, origin_Node: int, x_vertex_node: int, y_vertex_node: int, vertices_list: list[int], auto_generate: bool*) [source]

Define a parametric surface.

Parameters:

- **name** (*str*) – Name of the parametric surface.
- **type** (*int*) –

  Type of the parametric surface:

value	Surface Type
0	None
1	Wall
2	Slab

- **origin_Node** (*int*) – Node number defining the origin of the parametric surface.
- **x_vertex_node** (*int*) – Node number defining the local X axis of the parametric surface.
- **y_vertex_node** (*int*) – Node number defining the local Y axis of the parametric surface.
- **vertices_list** (*list[int]*) – List of vertices of the parametric surface. (must lie in same plane)
- **auto_generate** (*bool*) – Specifies whether to auto-generate boundary points and density objects for the parametric surface (True = Auto-generate, False = Otherwise)

Returns:

The ID of the created parametric surface.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> surface_id = staad_obj.Geometry.DefineParametricSurface("Surface1",
>>> print(surface_id)
```

DeleteBeam(*BeamNo: int*) [source]

Delete a specified beam.

Parameters:

BeamNo (*int*) – Beam number to delete.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.DeleteBeam(5)
```

DeleteGroup(*groupName: str*) [source]

Delete a group.

Parameters:

groupName (*str*)

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.DeleteGroup("GroupA")
```

DeleteNode(*nNodeNo: int*) [source]

Delete a specified node.

Parameters:

nNodeNo (*int*) – Node number to delete.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.DeleteNode(10)
```

`DeletePhysicalMember(physicalMemberId: int)` [source]

Delete a physical member.

Parameters:

physicalMemberId (*int*) – ID of the physical member to delete.

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.DeletePhysicalMember(1)
>>> print(result)
```

`DeletePlate(nPlateNo: int)` [source]

Delete a specified plate.

Parameters:

nPlateNo (*int*) – Plate number to delete.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.DeletePlate(1)
```

DeleteSolid(*solidID*) [source]

Delete a specified solid.

Parameters:

solidID (*int*) – ID of solid to delete.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.DeleteSolid(1)
```

DoTranslationalRepeat(*link_bays: bool, open_base: bool, axis_dir: int, spacing_list: list[float], no_of_bays: int, renumber_bays: bool, renumber_list: list[int], geometry_only_flag: bool*) [source]

Perform a translational repeat operation.

Parameters:

- **link_bays** (*bool*) – specifies whether to generate new members between each step in the direction of the repeat (True = Link Bays, False = Otherwise)
- **open_base** (*bool*) – specifies not to generate linking members at the base of the structure (i.e., the lowest nodes in the selection) (True = Open base, False = Otherwise)
- **axis_dir** (*int*) – value to specify direction in global axis along which translational repeat operation is to be performed (0 = GX, 1 = GY, 2 = GZ)

- **spacing_list** (*list[float]*) – List of spacing distances.
- **no_of_bays** (*int*) – specifies number of generated bays (maximum no of bays that can be generated single call of the API = 100)
- **renumber_bays** (*bool*) – specifies whether to use a user-specified starting number of the members generated in each newly generated bay (True = Renumber, False = Otherwise)
- **renumber_list** (*list[int]*) – specify starting member numbers for each newly generated bays (length of list = no_of_bays). if renumber_bays = False, this parameter is ignored.
- **geometry_only_flag** (*bool*) – specifies whether only geometry data is to be copied (True = Copy geometry only, False = Copy all)

Returns:

Result

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.DoTranslationalRepeat(True, True, 3, [1
>>> print(result) # True if successful, False otherwise
```

GetAnalyticalMemberCountForPhysicalMember(*physicalMemberId: int*) [source]

Get the count of analytical members in a physical member.

Parameters:

physicalMemberId (*int*) – ID of the physical member.

Returns:

Count of analytical members in the physical member.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetAnalyticalMemberCountForPhysicalMembe
>>> print(count)
```

GetAnalyticalMembersForPhysicalMember(*physicalMemberId: int*)

[source]

Get the analytical members in a physical member.

Parameters:

physicalMemberId (*int*) – ID of the physical member.

Returns:

List of analytical member IDs in the physical member.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> members = staad_obj.Geometry.GetAnalyticalMembersForPhysicalMember(
>>> print(members)
```

GetAreaOfPlates(*plateList*)

[source]

Get the area of plates.

Parameters:

plateList (*list of int*)

Returns:

list of area of each plate in the list.

Return type:

List

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> area_list = staad_obj.Geometry.GetAreaOfPlates([1,2,3])
>>> print(area_list)
```

`GetBeamLength(beam: int)` [source]

Get the length of a beam.

Parameters:

beam (*int*) – Beam number.

Returns:

Length of the beam.

Return type:

float

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> length = staad_obj.Geometry.GetBeamLength(1)
>>> print(length)
```

`GetBeamList()` [source]

Get the list of all beam numbers.

Returns:

List of beam numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> beams = staad_obj.Geometry.GetBeamList()
>>> print(beams)
```

GetBeamsConnectedAtNode(*node*) [source]

Get the list of beams connected at a node.

Parameters:

node (*int*) – Node number.

Returns:

Beam numbers connected at the node.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> beams = staad_obj.Geometry.GetBeamsConnectedAtNode(1)
>>> print(beams)
```

GetCountOfBreakableBeamsAtSpecificNodes(*nodeList: list*) [source]

Get number of beams that can be broken based on the list of node Ids.

Parameters:

nodeList (*list of int*) – List of node IDs to check for breakable beams.

Returns:

Count of breakable beams.

Return type:

int

See also

BreakBeamsAtSpecificNodes

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetCountOfBreakableBeamsAtSpecificNodes(
>>> print(count)
```

GetFlagForHiddenEntities() [source]

Get the flag specified for consideration of hidden entities (nodes and plates) while getting count or list of those entities

Returns:

All entities = 0 (Default option), Ignore Hidden entities = 1, Only hidden entities = 2

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> flag = staad_obj.Geometry.GetFlagForHiddenEntities()
>>> print(flag)
```

GetGeneratedQuadPanelIncidences() [source]

Get the incidences of generated quad panels for selected beams.

Returns:

List of 4 lists containing NodeAs, NodeBs, NodeCs, NodeDs in respective order. [[A1, A2, ...], [B1, B2, ...], [C1, C2, ...], [D1, D2, ...]]

Return type:

List of lists of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> incidences = staad_obj.Geometry.GetGeneratedQuadPanelIncidences(1)
```

`GetGroupCount(grouptype)` [source]

Get the number of groups of a given type.

Parameters:

grouptype (*int*) –

Index	Group Type
1	Nodes
2	Members
3	Plates
4	Solids
5	Geometry (Members, Plates and Solids)
6	Floor (Floor beam)

Returns:

Number of groups.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetGroupCount(1) # For Node groups
>>> print(count)
```

GetGroupCountAll() [source]

Get the total number of groups.

Returns:

Total group count.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> total_groups = staad_obj.Geometry.GetGroupCountAll()
>>> print(total_groups)
```

GetGroupEntities(*group_name*) [source]

Get the list of entities in a group.

Parameters:

group_name (*str*) – Name of the group.

Returns:

Entity numbers in the group.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> entities = staad_obj.Geometry.GetGroupEntities("Group1")
>>> print(entities)
```

`GetGroupEntityCount(group_name)` [source]

Get the number of entities in a group.

Parameters:

group_name (*str*) – Name of the group.

Returns:

Number of entities in the group.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetGroupEntityCount("Group1")
>>> print(count)
```

`GetGroupNames(grouptype)` [source]

Get the names of all groups of a given type.

Parameters:

grouptype (*int*) –

Index	Group Type
1	Nodes
2	Members
3	Plates
4	Solids
5	Geometry (Members, Plates and Solids)
6	Floor (Floor beam)

Returns:

List of group names.

Return type:

list of str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> names = staad_obj.Geometry.GetGroupNames(1)
>>> print(names)
```

GetIntersectBeamsCount(*beamList: list, tolerance: float*) [source]

Get the count of intersecting beams.

Parameters:

- **beamList** (*list of int*) – list of beam IDs to check for intersection. if it is empty, all beams in the model are considered.
- **tolerance** (*float*) – Tolerance to be used for finding beam intersection, should not be negative value, meter for Metric and inch for English in Base Units.

Returns:

Number of intersecting beams.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetIntersectBeamsCount([1,2,3])
```

GetLastBeamNo() [source]

Get the last beam ID.

Returns:

Last beam number.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> last_beam = staad_obj.Geometry.GetLastBeamNo()
>>> print(last_beam)
```

GetLastNodeNo() [source]

Get the last node number.

Returns:

The last node number. - 1 : General error.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> last_node = staad_obj.Geometry.GetLastNodeNo()
>>> print(last_node)
```

`GetLastPhysicalMemberNo()` [source]

Get the last physical member number.

Returns:

Last physical member number.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> last_no = staad_obj.Geometry.GetLastPhysicalMemberNo()
>>> print(last_no)
```

`GetLastPlateNo()` [source]

Get the last plate number.

Returns:

Last plate number.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> last_plate = staad_obj.Geometry.GetLastPlateNo()
>>> print(last_plate)
```

`GetLastSolidNo()` [source]

Returns the solid number of the last solid created in the model.

Returns:

Last solid number.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> last_solid = staad_obj.Geometry.GetLastSolidNo()
```

`GetMemberCount()` [source]

Get number of beam.

Returns:

Number of beams.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> beam_count = staad_obj.Geometry.GetMemberCount()
>>> print(beam_count)
```

`GetMemberIncidence(`*beam*`)` [source]

Get the start and end node numbers of a beam.

Parameters:

beam (*int*) – Beam number.

Returns:

(start_node, end_node)

Return type:

tuple of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> start, end = staad_obj.Geometry.GetMemberIncidence(1)
>>> print(f"Start Node: {start}, End Node: {end}")
```

GetMemberIncidence_CIS2(*memberId: int*) [source]

Get the incidence of a member in CIS/2 format.

Parameters:

memberId (*int*) – ID of the member.

Returns:

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

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> incidence = staad_obj.Geometry.GetMemberIncidence_CIS2(1)
>>> print(incidence)
```

GetMemberUniqueID(*memberNo: int*) [source]

Get the unique ID of a member.

Parameters:

memberNo (*int*)

Returns:

Unique ID of the member.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetMemberUniqueID(1)
>>> print(uid)
```

GetNoOfBeamsConnectedAtNode(*node*) [source]

Get the number of beams connected at a node.

Parameters:

node (*int*) – Node number.

Returns:

Number of beams connected at the node.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> connected_beam_count = staad_obj.Geometry.GetNoOfBeamsConnectedAtNo
>>> print(connected_beam_count)
```

GetNoOfGeneratedQuadPanels() [source]

Get the number of generated quad panels for selected beams.

Returns:

Number of generated quad panels.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> quad_panel_count = staad_obj.Geometry.GetNoOfGeneratedQuadPanels()
>>> print(quad_panel_count)
```

GetNoOfSelectedBeams()

[source]

Get the number of selected beams.

Returns:

Number of selected beams.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> selected_beam_count = staad_obj.Geometry.GetNoOfSelectedBeams()
>>> print(selected_beam_count)
```

GetNoOfSelectedNodes()

[source]

Get the number of selected nodes.

Returns:

Number of selected nodes.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> no_of_selected_nodes = staad_obj.Geometry.GetNoOfSelectedNodes()
>>> print(no_of_selected_nodes)
```

`GetNoOfSelectedPhysicalMembers()` [source]

Get the number of selected physical members.

Returns:

Number of selected physical members.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetNoOfSelectedPhysicalMembers()
>>> print(count)
```

`GetNoOfSelectedPlates()` [source]

Return the number of selected plates.

Returns:

Number of selected plates.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> selected_plate_count = staad_obj.Geometry.GetNoOfSelectedPlates()
>>> print(selected_plate_count)
```

GetNoOfSelectedSolids() [source]

Get the number of selected solids.

Returns:

Number of selected solids.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> n_solids = staad_obj.Geometry.GetNoOfSelectedSolids()
>>> print(n_solids)
```

GetNodeCoordinates(*node: int*) [source]

Get the coordinates of a node.

Parameters:

node (*int*) – Node number.

Returns:

(x, y, z) coordinates.

Return type:

tuple of float

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> coords = staad_obj.Geometry.GetNodeCoordinates(1)
>>> print(coords)
```

GetNodeCount() [source]

Get the total number of nodes.

Returns:

Number of nodes.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetNodeCount()
>>> print(count)
```

GetNodeDistance(*nodeA*, *nodeB*) [source]

Get the distance between two nodes.

Parameters:

- **nodeA** (*int*) – First node number.
- **nodeB** (*int*) – Second node number.

Returns:

Distance between the nodes.

Return type:

float

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> dist = staad_obj.Geometry.GetNodeDistance(1, 2)
>>> print(dist)
```

GetNodeIncidence(*node*) [source]

Get the incidence (coordinates) of a node.

Parameters:

node (*int*) – Node number.

Returns:

(x, y, z) Coordinates of the node.

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> inc = staad_obj.Geometry.GetNodeIncidence(1)
>>> print(inc)
```

GetNodeIncidence_CIS2(*nodeId: int*) [source]

Get the incidence of a node in CIS/2 format.

Parameters:

nodeId (*int*) – ID of the node.

Returns:

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

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> incidence = staad_obj.Geometry.GetNodeIncidence_CIS2(1)
>>> print(incidence)
```

`GetNodeList()` [source]

Get the list of all node numbers.

Returns:

List of node numbers.

Return type:

list

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> nodes = staad_obj.Geometry.GetNodeList()
>>> print(nodes)
```

`GetNodeNumber(x_y_z_coordinates: tuple)` [source]

Get the node number from coordinates.

Parameters:

x_y_z_coordinates (*tuple of float*) – (x, y, z) coordinates.

Returns:

Node number.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> node_no = staad_obj.Geometry.GetNodeNumber((0.0, 0.0, 0.0))
>>> print(node_no)
```

GetNodeUniqueID(*nodeNo: int*) [source]

Get the unique ID of a node.

Parameters:

nodeNo (*int*)

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetNodeUniqueID(1)
>>> print(uid)
```

GetPID(*EntityNo: int, EntityType: int*) [source]

Get the property ID of a member.

Parameters:

- **EntityNo** (*int*) – ID of the entity.
- **EntityType** (*int*) – Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Solid, 5 for Surface).

Returns:

Property ID of the member.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> pid = staad_obj.Geometry.GetPID(1)
>>> print(pid)
```

GetPMemberCount() [source]

Get the count of physical members in the model.

Returns:

Count of physical members.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetPMemberCount()
>>> print(count)
```

GetParametricSurfaceCount() [source]

Get the number of parametric surfaces.

Returns:

Number of parametric surfaces.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> num_surfaces = staad_obj.Geometry.GetParametricSurfaceCount()
```

GetParametricSurfaceInfo(*surfaceNo: int*) [source]

Get extended information about a parametric surface.

Parameters:

surfaceNo (*int*)

Returns:

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

Return type:

tuple of various details about the surface

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> surface_name, surface_type, boundary_points, density_points, openin
```

GetParametricSurfaceInfoEx(*surfaceNo: int*) [source]

Get information about a parametric surface.

Parameters:

surfaceNo (*int*)

Returns:

1. Surface Name (str) : Name of the Mesh
2. Surface Type (int) : (0: None, 1: Wall, 2: Slab)
3. Surface sub-type (str) : Sub type of the surface
4. number of vertices (int) : Number of vertices after meshing
5. Mesh Size (float) : Target mesh size

6. Divisions (int) : Number of divisions along the boundary
7. Meshing method (int) : (0: Basic, 1: Advanced)
8. isQuad (bool) : Whether the mesh is Quad or Triangular (True = Quad, False = Triangular)
9. Origin Node (int) : Origin Node ID
10. X Node (int) : Node ID on X axis to determine x axis
11. Y Node (int) : Node ID towards positive Y axis
12. Number of Circular Openings (int) : Number of circular openings
13. Number of Polygonal Openings (int) : Number of polygonal openings
14. Number of Circular Regions (int) : Number of circular regions
15. Number of Polygonal Regions (int) : Number of polygonal regions
16. Number of Density Points (int) : Number of density points
17. Number of Density Lines (int) : Number of density lines

Return type:

tuple containing various details about the surface

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> surface_name, surface_type, surface_sub_type, number_of_vertices, m
```

GetParametricSurfaceMeshData(*surfaceNo: int*) [source]

Gets data about specified parametric surface available in the currently loaded model

Parameters:

surfaceNo (*int*)

Returns:

1. Nodes (list) : List of generated node ids
2. Elements (list) : List of generated element(plate) ids

Return type:

tuple containing mesh data

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> mesh_data = staad_obj.Geometry.GetParametricSurfaceMeshData(1)
>>> print(mesh_data)
```

GetParametricSurfaceMeshInfo(*surfaceNo: int*) [source]

Gets information about specified parametric surface available in the currently loaded model.

Parameters:

surfaceNo (*int*) – Surface ID of the parametric surface.

Returns:

1. Node count (int) : Number of nodes in the mesh
2. Element count (int) : Number of elements in the mesh

Return type:

tuple containing mesh details

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> nodes, elements = staad_obj.Geometry.GetParametricSurfaceMeshInfo(1
>>> print(f"Nodes: {nodes}, Elements: {elements}")
```

GetParametricSurfaceSubType(*surfaceName: str*) [source]

Get the subtype of a parametric surface.

Parameters:

surfaceNo (*int*)

Returns:

str

Return type:

subtype information

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> subtype = staad_obj.Geometry.GetParametricSurfaceSubType("SECOND_FL
>>> print(subtype)
```

GetParametricSurfaceUniqueID(*surface_name: str*) [source]

Get the unique ID of a parametric surface.

Parameters:

surface_name (*str*)

Returns:

Unique ID of the parametric surface.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetParametricSurfaceUniqueID("SECOND_FLOOR
>>> print(uid)
```

GetPhysicalMemberCount() [source]

Get the count of physical members in the model.

Returns:

Count of physical members.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> count = staad_obj.Geometry.GetPhysicalMemberCount()
>>> print(count)
```

GetPhysicalMemberList() [source]

Get the list of physical members in the model.

Returns:

List of physical member IDs.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> members = staad_obj.Geometry.GetPhysicalMemberList()
>>> print(members)
```

GetPhysicalMemberUniqueID(*physicalMemberId: int*) [source]

Get the unique ID of a physical member.

Parameters:

physicalMemberId (*int*) – ID of the physical member.

Returns:

Unique ID of the physical member.

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetPhysicalMemberUniqueID(1)
>>> print(uid)
```

GetPlateCount() [source]

Returns the number of plates.

Returns:

Number of plates.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> plate_count = staad_obj.Geometry.GetPlateCount()
>>> print(plate_count)
```

GetPlateIncidence(*plateNo: int*) [source]

Get the node incidences A, B, C, D for a plate.

Parameters:

plateNo (*int*) – Plate number.

Returns:

4 end node IDs for the plate (A, B, C, D).

Return type:

tuple of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> a, b, c, d = staad_obj.Geometry.GetPlateIncidence(1)
>>> print(a, b, c, d)
```

GetPlateIncidence_CIS2(*plateId: int*) [source]

Get the incidence of a plate in CIS/2 format.

Parameters:

plateId (*int*) – ID of the plate.

Returns:

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

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> incidence = staad_obj.Geometry.GetPlateIncidence_CIS2(1)
>>> print(incidence)
```

GetPlateList() [source]

Returns the list of all plate numbers.

Returns:

List of plate numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> plates = staad_obj.Geometry.GetPlateList()
>>> print(plates)
```

`GetPlateNodeCount(plateNo: int)` [source]

Get the number of nodes in a plate.

Parameters:

plateNo (*int*) – Plate number.

Returns:

Number of nodes in the plate.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> node_count = staad_obj.Geometry.GetPlateNodeCount(1)
>>> print(node_count)
```

`GetPlateUniqueID(plateNo: int)` [source]

Get the unique ID of a plate.

Parameters:

plateNo (*int*)

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetPlateUniqueID(1)
>>> print(uid)
```

GetSelectedBeams() [source]

Get the list of selected beam numbers.

Returns:

Selected beam numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> selected = staad_obj.Geometry.GetSelectedBeams()
>>> print(selected)
```

GetSelectedNodes() [source]

Get the list of selected node numbers.

Returns:

Selected node numbers.

Return type:

list

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> selected = staad_obj.Geometry.GetSelectedNodes()
>>> print(selected)
```

GetSelectedPhysicalMembers() [source]

Get the list of selected physical members.

Returns:

List of selected physical member IDs.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> members = staad_obj.Geometry.GetSelectedPhysicalMembers()
>>> print(members)
```

GetSelectedPlates(*isSorted: bool = False*) [source]

return a list of selected plate numbers.

Parameters:

isSorted (*bool optional*) – If True, the plate numbers will be sorted. The default is False. (in the selected order)

Returns:

Selected plate numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> plates = staad_obj.Geometry.GetSelectedPlates(True)
>>> print(plates)
```

GetSelectedSolids(*isSorted: bool = False*) [source]

Get the list of selected solid numbers.

Parameters:

isSorted (*bool optional*) – If True, the solid numbers will be sorted. The default is False. (in the selected order)

Returns:

Selected solid numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> solids = staad_obj.Geometry.GetSelectedSolids(True)
>>> print(solids)
```

`GetSolidCount()` [source]

Returns the number of solids.

Returns:

Number of solids.

Return type:

int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> solid_count = staad_obj.Geometry.GetSolidCount()
>>> print(solid_count)
```

`GetSolidIncidence(`*`solidNo`*`)` [source]

Get the node incidences for a solid.

Returns:

8 end node IDs for the solid. (A, B, C, D, E, F, G, H)

Return type:

tuple of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> nodes = staad_obj.Geometry.GetSolidIncidence(1)
>>> print(nodes)
```

GetSolidIncidence_CIS2(*solidId: int*) [source]

Get the incidence of a solid in CIS/2 format.

Parameters:

solidId (*int*) – ID of the solid.

Returns:

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

Return type:

tuple

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> incidence = staad_obj.Geometry.GetSolidIncidence_CIS2(1)
>>> print(incidence)
```

GetSolidList() [source]

Get the list of all solid numbers.

Returns:

List of solid numbers.

Return type:

list of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> solids = staad_obj.Geometry.GetSolidList()
>>> print(solids)
```

GetSolidUniqueID(*solidNo: int*) [source]

Get the unique ID of a solid.

Parameters:

solidNo (*int*)

Return type:

str

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> uid = staad_obj.Geometry.GetSolidUniqueID(1)
>>> print(uid)
```

IntersectBeams(*method: int, beamList: list, tolerance: float*) [source]

Intersect beams.

Parameters:

- **method** (*int*) –

Index	Method
1	Highlight
2	Intersect

- **beamList** (*list of int*) – list of beam IDs to intersect. if it is empty, all beams in the model are considered.
- **tolerance** (*float*) – Tolerance to be used for finding beam intersection, should not be negative value, meter for Metric and inch for English in Base Units.

Returns:

IDs of the beams that have been changed and added, only used for intersect method.

Return type:

List of int

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> new_Ids = staad_obj.Geometry.IntersectBeams([1,2,3])
>>> print(new_Ids)
```

IsBeam(*beam_no: int, tol_angle: float*) [source]

Returns True if the angle of inclination for specified BEAM member is not more than given tolerance angle (for small angle only).

Parameters:

- **beam_no** (*int*)
- **tol_angle** (*float*)

Returns:

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

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> is_beam = staad_obj.Geometry.IsBeam(1, 1)
>>> print(is_beam)
```

IsColumn(*column_no: int, tol_angle: float*) [source]

Returns True if the angle of inclination for specified COLUMN member is not more than given tolerance angle (for small angle only).

Parameters:

- **column_no** (*int*)
- **tol_angle** (*float*)

Returns:

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

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> is_column = staad_obj.Geometry.IsColumn(1, 1)
>>> print(is_column)
```

IsOrphanNode(*nodeNo: int*) [source]

Check if a node is an orphan.

Parameters:

nodeNo (*int*)

Returns:

True if orphan, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> is_orphan = staad_obj.Geometry.IsOrphanNode(1)
>>> print(is_orphan)
```

`IsZUp()` [source]

Check if the Z axis is up.

Returns:

True if Z is up, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> is_z_up = staad_obj.Geometry.IsZUp()
>>> print(is_z_up)
```

`MergeBeams(beamList: list, newId: int, property_id: int, beta_angle: float, material_name: str)` [source]

Merge beams.

Parameters:

- **beamList** (*list of int*) – List of beam IDs to merge.
- **newId** (*int*) – New ID for the merged beam.
- **property_id** (*int*) – Property ID to assign to the merged beam.

- **beta_angle** (*float*) – Beta angle for the merged beam.
- **material_name** (*str*) – Material name for the merged beam.

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.MergeBeams([1,2,3], 4, 5, 30.0, "Steel"
>>> print(result)
```

MergeNodes(*new_Id: int, nodeList: list*) [source]

Merge nodes.

Parameters:

- **new_Id** (*int*) – New ID for the merged node.
- **nodeList** (*list of int*) – List of node IDs to merge.

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.MergeNodes(4, [1,2,3])
>>> print(result)
```

RemoveParametricSurfaceMesh(*surfaceNo: int*) [source]

Remove the specified parametric mesh from the model.

Parameters:

surfaceNo (*int*) – surface ID of the parametric surface to be delete from model.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.RemoveParametricSurfaceMesh(1)
>>> print(result)
```

`RenumberBeam(oldBeamNo: int, newBeamNo: int)` [source]

Renumber a beam.

Parameters:

- **oldBeamNo** (*int*)
- **newBeamNo** (*int*)

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.RenumberBeam(1, 10)
>>> print(result)
```

`SelectBeam(beamID)` [source]

Select a beam.

Parameters:

beamID (*int*) – beam number to select.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.SelectBeam(1)
>>> print(result)
```

SelectMultipleBeams(*beam_ids: list*) [source]

Select multiple beams.

Parameters:

beam_ids (*list of int*) – List of beam numbers to select.

Return type:

None

Examples

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

SelectMultipleNodes(*nodes: list*) [source]

Select multiple nodes.

Parameters:

nodes (*list*) – node numbers to select.

Return type:

bool

Examples

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

`SelectMultiplePhysicalMembers(physicalMemberList: list)` [source]

Select multiple physical members.

Parameters:

physicalMemberList (*list of int*) – List of physical member IDs to select.

Return type:

None

Examples

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

`SelectMultiplePlates(plates: list)` [source]

Select multiple plates.

Parameters:

plates (*list*) – Plate numbers to select.

Return type:

bool

Examples

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

SelectMultipleSolids(*solids: list*) [source]

Select multiple solids.

Parameters:

solids (*list*) – Solid numbers to select.

Return type:

bool

Examples

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

SelectNode(*nodeID*) [source]

Select a node.

Parameters:

nodeID (*int*) – node number to select.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.SelectNode(1)
>>> print(result)
```

SelectPhysicalMember(*physicalMemberId: int*) [source]

Select a physical member.

Parameters:

physicalMemberId (*int*) – ID of the physical member to select.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SelectPhysicalMember(1)
```

SelectPlate(*nPlateNo: int*) [source]

Select a plate by its number.

Parameters:

nPlateNo (*int*) – Plate number to select.

Returns:

Status of selection

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.SelectPlate(1)
>>> print(result) # True if selected, False otherwise
```

SelectSolid(*solidID*) [source]

Select a solid.

Parameters:

solidID (*int*) – solid number to select.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> result = staad_obj.Geometry.SelectSolid(1)
>>> print(result)
```

`SetCheckForIdenticalEntity(entityType: int, checkFlag: bool)` [source]

This API will set whether to enable checking for existing identical entities (beam, plate, node etc.) or not. If set is enabled, time taken by the corresponding add/create multiple entities APIs will take longer time, otherwise if set is disabled, time taken by corresponding APIs will be comparatively less.

Parameters:

- **entityType** (*int*) – Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Solid, 5 for Surface).
- **checkFlag** (*bool*) – Flag to check for identical entities.

Returns:

True if successful, False otherwise.

Return type:

bool

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetCheckForIdenticalEntity(1, True)
```

`SetFlagForHiddenEntities(flag: int)` [source]

Set the flag specified for consideration of hidden entities (nodes and plates) while getting count or list of those entities

Parameters:

flag (*int*) – All entities = 0 (Default option), Ignore Hidden entities = 1, Only hidden entities = 2

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetFlagForHiddenEntities(1)
```

SetMemberUniqueID(*beamNo: int, uniqueID: str*) [source]

Set a unique ID for a member.

Parameters:

- **beamNo** (*int*) – Beam number to set the unique ID for.
- **uniqueID** (*str*) – unique identifier for the member.

Return type:

None

Examples

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

SetNodeCoordinate(*nodeNo: int, x: float, y: float, z: float*) [source]

Set the coordinates of a node.

Parameters:

- **nodeNo** (*int*)
- **x** (*float*)
- **y** (*float*)
- **z** (*float*)

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetNodeCoordinate(1, 0.0, 0.0, 0.0)
```

SetNodeUniqueID(*nodeNo: int, uniqueID: str*) [source]

Set a unique ID for a node.

Parameters:

- **nodeNo** (*int*) – Node number to set the unique ID for.
- **uniqueID** (*str*) – Unique identifier for the node.

Return type:

None

Examples

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

SetPID(*EntityNo: int, EntityType: int, PropertyID: int*) [source]

Set the property ID of a member.

Parameters:

- **EntityNo** (*int*) – ID of the entity.

- **EntityType** (*int*) – Type of the entity. (1 for Node, 2 for Beam, 3 for Plate, 4 for Solid, 5 for Surface).
- **PropertyID** (*int*) – Property ID to set for the member.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetPID(1, 2, 5)
```

SetParametricSurfaceSubType(*surfaceName: str, subType: str*)

Set the subtype for a parametric surface. [source]

Parameters:

- **surfaceName** (*str*) – Name of the parametric surface.
- **subType** (*str*) – Sub-type of surface.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetParametricSurfaceSubType("SECOND_FLOOR_SLAB",
```

SetParametricSurfaceUniqueID(*surface_name: str, unique_id: str*)

Set a unique ID for a parametric surface. [source]

Parameters:

- **surface_name** (*str*)
- **unique_id** (*str*)

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SetParametricSurfaceUniqueID("SECOND_FLOOR_SLAB"
```

SetPhysicalMemberUniqueID(*physicalMemberId: int, uniqueId: str*)

[source]

Set the unique ID for a physical member.

Parameters:

- **physicalMemberId** (*int*) – ID of the physical member.
- **uniqueId** (*str*) – Unique ID to set for the physical member.

Return type:

None

Examples

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

SetPlateUniqueID(*plateNo: int, uniqueID: str*)

[source]

Set a unique ID for a plate.

Parameters:

- **plateNo** (*int*) – plate number to set the unique ID for.
- **uniqueID** (*str*) – unique identifier for the plate.

Return type:

None

Examples

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

SetSolidUniqueID(*solidNo: int, uniqueID: str*) [source]

Set a unique ID for a solid.

Parameters:

- **solidNo** (*int*) – Solid number to set the unique ID for.
- **uniqueID** (*str*) – Unique identifier for the solid.

Return type:

None

Examples

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

SplitBeam(*beamNo: int, nodes: int, distToNodes: list*) [source]

Split a beam into parts.

Parameters:

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

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SplitBeam(1, 2, [1.0, 2.0])
```

`SplitBeamInEqlParts`(*nBeamNo: int, nParts: int*) [source]

Split a beam into equal parts.

Parameters:

- **nBeamNo** (*int*) – Beam number to split.
- **nParts** (*int*) – Number of equal parts to split the beam into.

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.SplitBeamInEqlParts(1, 3)
```

`UpdateGroup`(*groupName: str, update_option: int, entityList: list[int]*) [source]

Updates (replaces, removes, adds) entities to a specified group.

Parameters:

- **groupName** (*str*)
- **update_option** (*int*) –

Index	Update Option
0	Replace entities
1	Remove entities
2	Add entities

- **entityList** (*list of int*)

Return type:

None

Examples

```
>>> from openstaadpy import os_analytical
>>> staad_obj = os_analytical.connect()
>>> staad_obj.Geometry.UpdateGroup("GroupA", [1,2,3])
```

`__init__`(*staadObj*) [source]