Oasys

Oasys GSA

API Reference

Oasys
YOUR IDEAS BROUGHT TO LIFE

8 Fitzroy Street
London
W1T 4BJ
Telephone: +44 (0) 20 7755 4515

Central Square
Forth Street
Newcastle Upon Tyne
NE1 3PL
Telephone: +44 (0) 191 238 7559

e-mail: oasys@arup.com
Website: oasys-software.com

Oasys GSA

Contents

Introduction

Gsa API is a dynamic linking library implemented in C++/CLI (Managed C++), which internally uses GSA C++ code to handle the data and commands. The dynamic linking library can be used in .Net applications, python by using Pythonnet package.

The API cannot be used in VBA scripts for Excel. But It shall be used to handle the data from Excel by using Microsoft.Office.Interop.Excel in a .Net application.

Getting started

.Net Application

Steps to use the API .Net application is given below.

Steps to use GsaAPI.dll in .Net Application:

- Install latest GSA 10.1 (for GsaAPI.dll)
- Add Gsa install path to System environment variable “Path” list (see the screen shot)
- The C# application that refers the GsaAPI.dll must be 64bit platform. As the API dll is 64bit, it cannot be used in AnyCPU/32bit platforms

Python

Steps to use the API in Python application is given below.

Steps to use GsaAPI.dll in python:

- Install pythonnet 2.4.0 (pip install pythonnet==2.4.0) https://pypi.org/project/pythonnet/2.4.0/. Make sure that >>> **import clr** works in python.
- Install latest GSA 10.1 (for GsaAPI.dll)
- Add Gsa install path to System environment variable “Path” list (see the screen shot)

Data Classes

Model is the main class that associates with Gsa model file. Data and results can be extracted from an opened model.

Model

```
{
	//Model handling related functions:
	//Parameters: filename – full path including folders
	Model();
	Model(string fileName);
	Model Clone(Model source);
	ReturnValue Open(string fileName);
	ReturnValue Close();
	ReturnValue Save();
	ReturnValue SaveAs(string fileName);



	//Analysis and Results related functions:
	//Parameters: analysisTaskID – Analysis Task number; analysisCaseID – Analysis
	Case number. Analysis case name is like “Analysis Case 1” and description is
	like “L1 + L2”.
	//Note: Add or Delete analysis case functions are supported on analysis tasks
	with solution type Static, Static P-Delta, Static NL.
	ReadOnlyDictionary<int, AnalysisCaseResult> Results();
	bool Analyse(int analysisTaskID);
	bool DeleteResults (int analysisTaskID);

	ReadOnlyDictionary<int, AnalysisTask> AnalysisTasks();
	string	AnalysisCaseDescription(int analysisCaseID);
	string	AnalysisCaseName(int analysisCaseID);
	ReturnValue	AddAnalysisCaseToTask(int analysisTaskID,
						string name,
						string description);
	ReturnValue	DeleteAnalysisCaseFromTask(int analysisTaskID,
						int analyisCaseID);
	bool	SetAnalysisCaseDescription(int analysisCaseID,
						string description);
	int AddAnalysisTask(); //Note: Added task type is Linear Static
	int AddCombinationCase(string name, string description);
```

```
//Axes related functions:
//Parameters: axisID is Axis reference number
ReadOnlyDictionary<int, Axis> Axes();
int    AddAxis(Axis axis);
void   AddAxes(ReadOnlyCollection<Axis> axes);
void   DeleteAxes(ReadOnlyCollection<int> axisIDs);
void   DeleteAxes(int axisID, int numberOfAxes);
void   DeleteAxis(int axisID);


//Beam loads related functions:
//Parameters: beamLoadID is beam load reference number
ReadOnlyCollection<BeamLoad> BeamLoads();
int    AddBeamLoad(BeamLoad beamLoad);
void   AddBeamLoads(ReadOnlyCollection<BeamLoad> beamLoads);
void   DeleteBeamLoad(int beamLoadID);
void   DeleteBeamLoads(int beamLoadID, int numberOfBeamLoads);
void   SetBeamLoad(int beamLoadID, BeamLoad beamLoad);


//Elements related functions:
//Parameters: elementID is element reference number; elementsList is a string
to specify the list of elements like “1 to 10”, “PB1” or it can be a list name
like “roof elements”.
ReadOnlyDictionary<int, Element> Elements();
ReadOnlyDictionary<int, Element> Elements(string elementsList);
int    AddElement(Element element);
void   AddElements(ReadOnlyCollection<Element> elements);
void   CreateElementsFromMembers();
void   DeleteElement(int elementID);
void   DeleteElements(string elementsList);
void   DeleteElements(ReadOnlyCollection<int> elementIDs);
void   SetElement(int elementID, Element element);
void   SetElements(ReadOnlyDictionary<int, Element> elements);
double ElementArea(int elementID);
ReadOnlyCollection<double> ElementDirectionCosine(int elementID);
double ElementLength(int elementID);
double ElementVolume(int elementID);

//Analysis Material related functions:
//Parameters: analysisMaterialID is analysis material reference number
```

```
ReadOnlyCollection<AnalysisMaterial>  AnalysisMaterials();  // Only elastic
isotropic
int   AddAnalysisMaterial(AnalysisMaterial material);
void  DeleteAnalysisMaterial(int analysisMaterialID);
void  SetAnalysisMaterial(int analysisMaterialID, AnalysisMaterial material);

//Face loads related functions:
//Parameters: faceLoadID is face load reference number
ReadOnlyCollection<FaceLoad> FaceLoads();
int   AddFaceLoad(FaceLoad faceLoad);
void  AddFaceLoads(ReadOnlyCollection<FaceLoad> faceLoads);
void  DeleteFaceLoad(int faceLoadID);
void  DeleteFaceLoads(int faceLoadID, int numberOfFaceLoads);
void  SetFaceLoad(int faceLoadID, FaceLoad faceLoad);


//Gravity loads related functions:
//Parameters: gravityLoadID is gravity load reference number
ReadOnlyCollection<GravityLoad> GravityLoads();
int   AddGravityLoad(GravityLoad gravityLoad);
void  AddGravityLoads(ReadOnlyCollection<GravityLoad> gravityLoads);
void  DeleteGravityLoad(int gravityLoadID);
void  DeleteGravityLoads(int gravityLoadID, int numberOfGravityLoads);
void  SetGravityLoad(int gravityLoadID, GravityLoad gravityLoad);


//Members related functions:
//Parameters: memberID is member reference number; memebersList is a string to
specify the list of members like “1 to 10” or it can be a list name like “Roof
members”.
ReadOnlyDictionary<int, Member> Members();
ReadOnlyDictionary<int, Member> Members(string membersList);
int   AddMember(Member member);
void  AddMembers(ReadOnlyCollection<Member> members);
void  DeleteMember(int memberID);
void  DeleteMembers(ReadOnlyCollection<int> memberIDs);
void  DeleteMembers(string membersList);
void  SetMember(int memberID, Member member);
void  SetMembers(ReadOnlyDictionary<int, Member> members);
double MemberArea(int memberID);
ReadOnlyCollection<double> MemberDirectionCosine(int memberID);
double MemberLength(int memberID);
```

```
//Node loads related functions:
//Parameters: nodeLoadID is node load reference number
ReadOnlyCollection<NodeLoad> NodeLoads(NodeLoadType type);
int    AddNodeLoad(NodeLoadType type, NodeLoad nodeLoad);
void   AddNodeLoads(NodeLoadType       type,       ReadOnlyCollection<NodeLoad>
nodeLoads);
void   DeleteNodeLoad(NodeLoadType type, int nodeLoadID);
void   DeleteNodeLoads(NodeLoadType      type,      int      nodeLoadID,      int
numberOfNodeLoads);
void   SetNodeLoad(NodeLoadType type, int nodeLoadID, NodeLoad nodeLoad);


//Nodes related functions:
//Parameters:  nodeID  is  node  reference  number;  nodesList  is  a  string  to
specify the list of nodes like “100 to 201” or it can a list name like “roof
nodes”.
ReadOnlyDictionary<int, Node> Nodes();
ReadOnlyDictionary<int, Node> Nodes(string nodesList);
int    AddNode(Node node);
void   AddNodes(ReadOnlyCollection<Node> nodes);
void   DeleteNode(int nodeID);
void   DeleteNodes(ReadOnlyCollection<int> nodeIDs);
void   DeleteNodes(string nodesList);
void   SetNode(int nodeID, Node node);
void   SetNodes(ReadOnlyDictionary<int, Node> nodes);


//2D property related functions:
//Parameters: prop2DID is 2D property reference number
ReadOnlyDictionary<int, Prop2D> Prop2Ds();
int    AddProp2D(Prop2D prop2D);
void   AddProp2Ds(ReadOnlyCollection<Prop2D> prop2Ds);
void   DeleteProp2D(int prop2DID);
void   DeleteProp2Ds(int prop2DID, int numberOfProp2Ds);
void   DeleteProp2Ds(ReadOnlyCollection<int> prop2DIDs);
void   SetProp2D(int prop2DID, Prop2D prop2D);
void   SetProp2Ds(ReadOnlyDictionary<int, Prop2D> prop2Ds);


//3D property related functions:
```

```
//Parameters: prop3DID is 3D property reference number
ReadOnlyDictionary<int, Prop3D> Prop3Ds();
int    AddProp3D(Prop3D prop3D);
void   AddProp3Ds(ReadOnlyCollection<Prop3D> prop3Ds);
void   DeleteProp3D(int prop3DID);
void   DeleteProp3Ds(int prop3DID, int numberOfProp3Ds);
void   DeleteProp3Ds(ReadOnlyCollection<int> prop3DIDs);
void   SetProp3D(int prop3DID, Prop3D prop3D);
void   SetProp3Ds(ReadOnlyDictionary<int, Prop3D> prop3Ds);


//Sections related functions:
//Parameters: sectionID is section reference number
ReadOnlyDictionary<int, Section> Sections();
int    AddSection(Section section);
void   AddSections(ReadOnlyCollection<Section> sections);
void   DeleteSection(int sectionID);
void   DeleteSections(int sectionID, int numberOfSections);
void   DeleteSections(ReadOnlyCollection<int> sectionIDs);
void   SetSection(int sectionID, Section section);
void   SetSections(ReadOnlyDictionary<int, Section> sections);


//Sections modifier related functions:
//Parameters: sectionModifierID is the reference number corresponding to the
modified section
ReadOnlyDictionary<int, SectionModifier> SectionModifiers();
int    AddSectionModifier(SectionModifier sectionModifier);
void   AddSectionModifiers(ReadOnlyCollection<SectionModifier>
sectionModifiers);
void   DeleteSectionModifier(int sectionModifierID);
void   DeleteSectionModifiers(int            sectionModifierID,            int
numberOfSectionModifiers);
void   DeleteSectionModifiers(ReadOnlyCollection<int> sectionModifierIDs);
void   SetSectionModifier(int        sectionModifierID,        SectionModifier
sectionModifier);
void   SetSectionModifiers(
            ReadOnlyDictionary<int, SectionModifier> sectionModifiers);


//Titles related functions:
```

```
        Titles Titles();
}
```

AnalysisMaterial

```
{
        //Properties
        double      CoefficientOfThermalExpansion   { get; set; }
        double      Density                         { get; set; }
        double      PoissonsRatio                         { get; set; }
        double      ShearModulus                    { get; set; }
        double      ElasticModulus                  { get; set; }
}
```

AnalysisTask

```
{
        //Properties
        int                         Type              { get; set; }
        ReadOnlyCollection<int>     Cases             { get; }
        ReadOnlyCollection<int>     TaskDependencies  { get; }
        string                      Name              { get; set; }
}
```

Axis

```
{
        //Properties
        AxisType     Type        { get;  set;}
        string       Name        { get;  set;}
        Vector3      Origin      { get;  set;}
        Vector3      XVector     { get;  set;}
        Vector3      XYPlane     { get;  set;}

        //Functions
        string TypeAsString();

}
```

BeamLoad

```
{
	//Properties
	BeamLoadType	Type		{ get; set; }
	bool		IsProjected	{ get; set; }
	Direction	Direction	{ get; set; }
	int		AxisProperty	{ get; set; }
	int		Case		{ get; set; }
	string		Elements	{ get; set; }
	string		Name		{ get; set; }

	//Functions
	Parameter: position is 0 based index.
	position = 0 represents Position 1
	position = 1 represents Position 2
	double		Position(int position);
	double		Value(int position);
	void		SetPosition(int position, double value);
	void		SetValue(int position, double value);
	string		DirectionAsString();
	string		TypeAsString();

	// Positions and values
	Position 1

	This is only relevant for point loads and patch loads. For a point load this
	is the location of the load measured from end 1 of the beam; for a patch load
	it is the start of the patch load measured from end 1 of the beam Position 1
	can be specified as a length or as a percentage of the length of the element,
	e.g. either 1.2 or 15%. As the variable is double, %values to be entered as
	negative values i.e. equivalent to 15% is -0.15

	Value 1

	Load magnitude. For a point load this is the load value; for a uniform load
	this is the load intensity; for a linear load this is the load intensity at
	end 1; for a patch load this is the load intensity at Position 1.

	Position 2
```

```
        This is only relevant for patch loads and is the end of the patch load
        measured from end 1 of the beam. Position 2 can be specified as a length or as
        a percentage of the length of the element, e.g. either 2.4 or 85%. As the
        variable is double, %values to be entered as negative values i.e. equivalent
        to 85% is -0.85


        Value 2

        Load magnitude. For a linear load this is the load intensity at end 2 and for
        a patch load this is the load intensity at Position 2.
}
```

Element

```
{
        //Properties
        bool          IsDummy             { get; set; }
        double        OrientationAngle    { get; set; }
        ElementType   Type                { get; set; }
        int           Group               { get; set; }
        int           OrientationNode     { get; set; }
        int           Property            { get; set; }
        Offset        Offset              { get; set; }
        ReadOnlyCollection<int> Topology      { get; set; }
        string        Name                { get; set; }
        ValueType     Colour              { get; set; }

        //Functions
        EndRelease    GetEndRelease(int iTopology);
        Bool6         Release(int iTopology);
        void          SetEndRelease(int iTopology, EndRelease endRelease);
        void          SetRelease(int iTopology, Bool6 release);
        string        TypeAsString();
}
```

FaceLoad

```
{
        //Properties
```

```
        bool        IsProjected  { get; set; }
        Direction   Direction    { get; set; }
        FaceLoadType       Type         { get; set; }
        int         AxisProperty { get; set; }
        int         Case         { get; set; }
        string      Elements     { get; set; }
        string      Name         { get; set; }
        Vector2     Position     { get; set; }

        //Functions
        double      Value(int position);
        void        SetValue(int position, double value);
        string      TypeAsString();

        // Variable – position
        Position (int) - varies from 1 to 4, value is the pressure corresponding to
        the position.

        Position (vector2) - Position of load is applicable for point load type. The
        position of the point load to be specified in ( r , s ) coordinates based on
        two-dimensional shape function. Coordinates vary from –1 to 1 for Quad4 and
        Quad8 and vary from 0 to 1 for Triangle3 and Triangle6.
```

Type	Direction	Pressure at Position				Position of Load	
		1	2	3	4	r	s
		[N/m² | N]	[N/m²]	[N/m²]	[N/m²]		
Uniform	z	0	0	0	0	0	0
Uniform	z	1.75					
Variable	z	1.5	1.25	1.4	1.82		
Point	z	20				-0.4	0.5

```
}
```

GravityLoad

```
{
        //Properties
        int         Case         { get; set; }
        string      Elements     { get; set; }
        string      Name         { get; set; }
        Vector3     Factor       { get; set; }
}
```

Member

```
{
	//Properties
	AnalysisOrder Type2D			{ get; set; }
	bool		IsDummy			{ get; set; }
	double		MeshSize		{ get; set; }
	double		OrientationAngle	{ get; set; }
	ElementType	Type1D			{ get; set; }
	int		Group			{ get; set; }
	int		MemberEnd1		{ get; set; }
	int		MemberEnd2		{ get; set; }
	int		OrientationNode		{ get; set; }
	int		Property		{ get; set; }
	MemberType	Type			{ get; set; }
	Offset		Offset			{ get; set; }
	string		Name			{ get; set; }
	string		Topology		{ get; set; }
	ValueType	Colour			{ get; set; }
	bool		IsIntersector			{ get; set; }
	Nullable<double>	LateralTorsionalBucklingFactor			{ get; set; }
	Nullable<double>	MomentAmplificationFactorStrongAxis		{ get; set; }
	Nullable<double>	MomentAmplificationFactorWeakAxis			{ get;
	set; }


	//Functions
	EndRelease	GetEndRelease(int iTopology);
	void		SetEndRelease(int iTopology, EndRelease endRelease);
	string		Type1DAsString();
	string		Type2DAsString();
	string		TypeAsString();
}
```

NodeLoad

```
{
	//Properties
	Direction	Direction	{ get; set; }
	double		Value		{ get; set; }
	int		AxisProperty	{ get; set; }
	int		Case		{ get; set; }
	string		Name		{ get; set; }
```

```
	string		Nodes		{ get; set; }
}
```

Node

```
{
	//Properties
	Bool6		Restraint		{ get; set; }
	int		AxisProperty			{ get; set; }
	int		DamperProperty		{ get; set; }
	int		MassProperty			{ get; set; }
	int		SpringProperty		{ get; set; }
	string		Name			{ get; set; }
	ValueType	Colour			{ get; set; }
	Vector3		Position		{ get; set; }
}
```

Prop2D

```
{
	//Properties
	double		Thickness			{ get; set; }
	int		AxisProperty				{ get; set; }
	int		MaterialAnalysisProperty	{ get; set; }
	int		MaterialGradeProperty		{ get; set; }
	MaterialType		MaterialType				{ get; set; }
	Property2D_Type	Type			{ get; set; }
	string		Name				{ get; set; }
	ValueType	Colour				{ get; set; }

	//Functions
	string		MaterialTypeAsString();
	string		TypeAsString();
}
```

Prop3D

```
{
	//Properties
	int		AxisProperty				{ get; set; }
	int		MaterialAnalysisProperty	{ get; set; }
```

```
        int          MaterialGradeProperty    { get; set; }
        MaterialType       MaterialType                { get; set; }
        string       Name                     { get; set; }
        ValueType    Colour                   { get; set; }

        //Functions
        string       MaterialTypeAsString();
}
```

Section

```
{
        //Properties
        double       Area                     { get; }
        double       Iyy                      { get; }
        double       Iyz                      { get; }
        double       Izz                      { get; }
        double       J                        { get; }
        double       Ky                       { get; }
        double       Kz                       { get; }
        double       SurfaceAreaPerLength     { get; }
        double       VolumePerLength          { get; }
        int          MaterialAnalysisProperty { get; set; }
        int          MaterialGradeProperty    { get; set; }
        int          Pool                     { get; set; }
        MaterialType       MaterialType               { get; set; }
        string       Name                     { get; set; }
        string       Profile                  { get; set; }
        ValueType    Colour                   { get; set; }

        //Functions
        string       MaterialTypeAsString();
}
```

SectionModifier

```
{
        //Properties
        double                   AdditionalMass          { get; set; }
        SectionModifierAttribute AreaModifier                  { get; set; }
        SectionModifierAttribute I11Modifier             { get; set; }
```

```
        SectionModifierAttribute I22Modifier              { get; set; }
        bool                     IsBendingAxesPrincipal   { get; set; }
        bool                     IsReferencePointCentroid { get; set; }
        SectionModifierAttribute JModifier                { get; set; }
        SectionModifierAttribute K11Modifier              { get; set; }
        SectionModifierAttribute K22Modifier              { get; set; }
        SectionModifierStressType       StressOption             { get; set; }
        SectionModifierAttribute VolumeModifier           { get; set; }
}
```

Titles

```
{
        //Properties
        string Calculation       { get; set; }
        string Initials    { get; set; }
        string JobNumber   { get; set; }
        string Notes             { get; set; }
        string SubTitle    { get; set; }
        string Title             { get; set; }
}
```

Result Classes

AnalysisCaseResult

```
{
	//Properties
	GlobalResult Global { get; }

	//Functions
	//Parameters: positions - can be in the range of 0.0 to 1.0, 0.5 is centre of the element); elementsList is a string to specify the list of elements like “1 to 10”, “PB1” or it can be a list name like “roof elements”.
	ReadOnlyDictionary<int, Element1DResult> Element1DResults(string elementsList,
		ReadOnlyCollection<double> positions);

	// positioncount - calculates the positions automatically, if positionCount is 5, positions are 0.0, 0.25, 0.50, 0.75, 1.0)
	ReadOnlyDictionary<int, Element1DResult> Element1DResults(string elementsList,
		int positionCount);

	// fLayer for 2D elements is -1, 0, 1 for bottom, middle, top
	ReadOnlyDictionary<int, Element2DResult> Element2DResults(string elementsList,
		double fLayer);

	ReadOnlyDictionary<int, Element3DResult> Element3DResults(string elementList);

	// nodesList is a string to specify the list of nodes like “100 to 201” or it can a list name like “roof nodes”.
	ReadOnlyDictionary<int, NodeResult> NodeResults(string nodesList);
}
```

CombinationCaseResult

```
{
	// These methods return a ReadOnlyCollection of results, one for each permutation of the combination case
	ReadOnlyDictionary<int, ReadOnlyCollection<Element1DResult>>
		Element1DResults(string elementsList, ReadOnlyCollection<double>
		positions);
```

```
	// positioncount - calculates the positions automatically, if positionCount is
	5, positions are 0.0, 0.25, 0.50, 0.75, 1.0)
	ReadOnlyDictionary<int,                    ReadOnlyCollection<Element1DResult>>
	            Element1DResults(string  elementsList,  int  positionCount,  bool
	            includeStrainEnergyResults = true);
	ReadOnlyDictionary<int,                    ReadOnlyCollection<Element2DResult>>
Element2DResults(string elementsList, double fLayer);
	ReadOnlyDictionary<int,                    ReadOnlyCollection<Element3DResult>>
Element3DResults(string elementsList);
	ReadOnlyDictionary<int,      ReadOnlyCollection<NodeResult>>NodeResults(string
nodesList);
}
```

GlobalResult

```
{
	//Properties
	double       Frequency;
	double       LoadFactor;
	double       ModalGeometricStiffness;
	double       ModalMass;
	double       ModalStiffness;
	Double6      TotalLoad;
	Double6      TotalReaction;
	int          Mode;
	Vector3      EffectiveInertia;
	Vector3      EffectiveMass;
}
```

Element1DResult

```
{
	//Properties
	ReadOnlyCollection<Double6>     Displacement;
	ReadOnlyCollection<Double6>     Force;
	ReadOnlyCollection<double>            StrainEnergyDensity;
	double                          AverageStrainEnergyDensity;
}
```

Element2DResult

```
{
	//Properties
	ReadOnlyCollection<Double6> Displacement;
	ReadOnlyCollection<Tensor2> Force;
	ReadOnlyCollection<Tensor2> Moment;
	ReadOnlyCollection<Vector2> Shear;
	ReadOnlyCollection<Tensor3> Stress;
}
```

Element3DResult

```
{
	//Properties
	ReadOnlyCollection<Double3> Displacement;
	ReadOnlyCollection<Tensor3> Stress;
}
```

NodeResult

```
{
	//Properties
	Double6 Constraint;
	Double6 Displacement;
	Double6 Reaction;
}
```

Types

Bool6

```
{
	//Properties
	bool X { get; }
	bool XX { get; }
	bool Y { get; }
	bool YY { get; }
	bool Z { get; }
	bool ZZ { get; }
}
```

Double3

```
{
	//Properties
	double X { get; }
	double Y { get; }
	double Z { get; }
}
```

Double6

```
{
	//Properties
	double X { get; }
	double XX { get; }
	double Y { get; }
	double YY { get; }
	double Z { get; }
	double ZZ { get; }
}
```

EndRelease

```
{
	//Properties
	//Fixed directions have null stiffnesses and vice versa
```

```
        Bool6 Releases { get; }
        NullableDouble6 Stiffnesses { get; }
}
```

NullableDouble6

```
{
        //Properties
        double? X { get; }
        double? XX { get; }
        double? Y { get; }
        double? YY { get; }
        double? Z { get; }
        double? ZZ { get; }
}
```

Offset

```
{
        //Properties
        double X1 { get; set; }
        double X2 { get; set; }
        double Y { get; set; }
        double Z { get; set; }
}
```

SectionModifierAttribute

```
{
        //Properties
        SectionModifierOptionTypeOption{ get; }
        Double                    Value { get; }
}
```

Tensor2

```
{
        //Properties
        double XX { get; }
        double XY { get; }
```

```
        double YX { get; }
        double YY { get; }
}
```

Tensor3

```
{
        //Properties
        double XX { get; }
        double XY { get; }
        double YY { get; }
        double YZ { get; }
        double ZX { get; }
        double ZZ { get; }
}
```

Vector2

```
{
        //Properties
        double X { get; }
        double Y { get; }
}
```

Vector3

```
{
        //Properties
        double X { get; set; }
        double Y { get; set; }
        double Z { get; set; }
}
```

Enums

AnalysisOrder

```
{
	LINEAR = 0,
	PARABOLIC = 1,
	RIGID_DIAPHRAGM = 2
}
```

AxisType

```
{
	OASYS_DATA_AXISTYPE_ENUMS = 0
}
```

BeamLoadType

```
{
	UNDEF = 0,
	POINT = 1,
	UNIFORM = 2,
	LINEAR = 3,
	PATCH = 4,
	TRILINEAR = 5
}
```

Direction

```
{
	NONE = 0,
	X = 1,
	Y = 2,
	Z = 3,
	XX = 4,
	YY = 5,
	ZZ = 6,
	XY = 7,
	YZ = 8,
	ZX = 9,
```

```
	YX = 10,
	ZY = 11,
	XZ = 12
}
```

ElementType

```
{
	NEW = -1,
	UNDEF = 0,
	FIRST_TYPE = 1,
	BAR = 1,
	BEAM = 2,
	SPRING = 3,
	MASS = 4,
	QUAD4 = 5,
	QUAD8 = 6,
	TRI3 = 7,
	TRI6 = 8,
	LINK = 9,
	CABLE = 10,
	TAPER_BEAM = 11,
	BRICK8 = 12,
	GRD_SPRING = 18,
	SPACER = 19,
	STRUT = 20,
	TIE = 21,
	BEAM3 = 22,
	ROD = 23,
	DAMPER = 24,
	GRD_DAMPER = 25,
	LAST_TYPE = 25,
	ZERO_D = 100,
	ONE_D = 101,
	TWO_D = 102,
	THREE_D = 103,
	ONE_D_SECT = 104,
	TWO_D_FE = 105,
	TWO_D_WALL = 106,
	TWO_D_LOAD = 107
}
```

FaceLoadType

```
{
	UNDEF = 0,
	CONSTANT = 1,
	GENERAL = 2,
	POINT = 3
}
```

MaterialType

```
{
	UNDEF = -2,
	NONE = -1,
	GENERIC = 0,
	STEEL = 1,
	FIRST = 1,
	CONCRETE = 2,
	ALUMINIUM = 3,
	GLASS = 4,
	FRP = 5,
	REBAR = 6,
	TIMBER = 7,
	FABRIC = 8,
	SOIL = 9,
	NUM_MT = 10,
	COMPOUND = 256,
	BAR = 4096,
	TENDON = 4352,
	FRPBAR = 4608,
	CFRP = 4864,
	GFRP = 5120,
	AFRP = 5376,
	ARGFRP = 5632,
	BARMAT = 65280
}
```

MemberType

```
{
```

```
	UNDEF = -1,
	GENERIC_1D = 0,
	GENERIC_2D = 1,
	BEAM = 2,
	COLUMN = 3,
	SLAB = 4,
	WALL = 5,
	CANTILEVER = 6,
	RIBSLAB = 7,
	COMPOS = 8,
	PILE = 9,
	EXPLICIT = 10,
	VOID_CUTTER_1D = 11,
	VOID_CUTTER_2D = 12
}
```

NodeLoadType

```
{
	NODE_LOAD = 0,
	APPLIED_DISP = 1,
	SETTLEMENT = 2,
	NUM_TYPES = 3
}
```

Property2D_Type

```
{
	UNDEF = 0,
	PL_STRESS = 1,
	PL_STRAIN = 2,
	AXISYMMETRIC = 3,
	FABRIC = 4,
	PLATE = 5,
	SHELL = 6,
	CURVED_SHELL = 7,
	TORSION = 8,
	WALL = 9,
	LOAD = 10
}
```

ReturnValue

```
{
	GS_OK = 0,
	GS_FILE_OPEN_ERROR = 1,
	GS_NO_RESULTS = 2,
	GS_NO_DATA_FOUND = 3,
	GS_FILE_SAVE_FAILED = 4,
	GS_UNSUPPORTED_FILE_FORMAT = 5,
	GS_SECTION_DB_MISSING = 6,
	GS_APPLICATION_AVF_MISSING = 7,
	GS_UNSUPPORTED_ANALYSIS_TASK_TYPE = 8,
	GS_ANALYSIS_TASK_DOESNT_EXIST = 9
}
```

SectionModifierOptionType

```
{
	BY = 1,
	TO = 2
}
```

SectionModifierStressType

```
{
	NO_MOD = 0,
	USE_UNMOD = 1,
	USE_MOD = 2
}
```

Miscellaneous

Data classes that are derived from SidAbility can get Sid string and set Sid tag to the data record. Axis, BeamLoad, Element, FaceLoad, GravityLoad, Member, Node, NodeLoad, Prop2D and Section objects can handle the Sid. Please refer GSA help file for more information about Sid.

SidAblility

```
{
	string		Sid { get; }
	void		SetSidTag(string tag, string value);
}
```

GsaApiException

```
{
	//Derived from System.Exception with the following additional properties
	string			ExtraInformation { get; }
	ApiExceptionCode	Code { get; }
}
```

ApiExceptionCode

```
{
	 GS_EX_GENERIC = 0,
	 GS_EX_NO_OPENFILE = 1,
	 GS_EX_INVALID_FILE_EXTENSION = 2,
	 GS_EX_UNSUPPORTED_NODE_TYPE = 3,
	 GS_EX_APPLICATION_AVF_MISSING = 4,
	 GS_EX_SECTION_DB_MISSING = 5,
	 GS_EX_UNSUPPORTED_ANALYSIS_TASK_TYPE = 6,
	 GS_EX_ANALYSIS_TASK_DOESNT_EXIST = 7,
	 GS_EX_API_EXPIRED = 8
}
```

Screen Shot: “Path” System Environment Variable

1. Go to Control Panel
2. Click System
3. On left hand pane click “Advanced system settings”
4. Make sure that the Advanced tab of system properties dialog is in view
5. Click “Environmental variables…” button
6. Go to System variable section (variable in bottom pane)
7. Open Path variable and add the path as shown in the following image

Note: If the permissions are not enough to update “system variables”, please update “Path” from “User variables”.