



COM interface

(AxisVM Library 2.0)

Reference Guide

Version 2.0a

COM interface	1
EGeneralError	4
IAxisVMApplication.....	5
IAxisVMCatalog	6
IAxisVMModels	7
IAxisVMModel.....	7
IAxisVMCalculation	9
IAxisVMCrossSections	10
IAxisVMCrossSection	20
IAxisVMDomains	22
IAxisVMDomain	24
IAxisVMLines	25
IAxisVMLine	27
IAxisVMLineSupports	30
IAxisVMLineSupport.....	32
IAxisVMLoadCases	33
IAxisVMLoadCombinations	34
IAxisVMLoadGroups.....	35
IAxisVMLoadGroup	36
IAxisVMLoads	37
IAxisVMMaterials	49
IAxisVMMaterial	55
IAxisVMNodalSupports	56
IAxisVMNodalSupport.....	58
IAxisVMNodes	59
IAxisVMReferences	61
IAxisVMSettings	62
IAxisVMSurfaces	63
IAxisVMSurface	64
IAxisVMSurfaceSupports.....	66
IAxisVMSurfaceSupport	67
IAxisVMLine2d	67
IAxisVMPolygon2d	68
IAxisVMPolygon2dList	68
IAxisVMLines3d.....	69
Event handling	71
IAxisVMApplicationEvents	71
IAxisVMModelsEvents	71
IAxisVMCrossSectionsEvents	71
IAxisVMDomainsEvents	71
IAxisVMLinesEvents	71
IAxisVMLineSupportsEvents	71
IAxisVMLoadCasesEvents.....	71
IAxisVMLoadCombinationsEvents	71
IAxisVMLoadGroupsEvents	71
IAxisVMLoadsEvents	72
IAxisVMMaterialsEvents	72
IAxisVMNodalSupportsEvents	72
IAxisVMNodesEvents	72
IAxisVMReferencesEvents	72
IAxisVMSettingsEvents.....	72
IAxisVMSurfacesEvents	72
IAxisVMSurfaceSupportsEvents	72
AxisVM COM plugin 1.0	73

Units

length	meter [m]
mass	kilogram [kg]
temperature	Celsius [°C]
time	second [s]
degree	radian [rad]
force	kilonewton [kN]

Other units are derived from these.

Data types

BOOL	logical value (= VARIANT_BOOL, <i>True</i> or <i>False</i>)
long	4 byte integer value
unsigned long	4 byte integer value
double	8 byte double precision floating point value
BSTR	double-byte Unicode string
Object*	4 byte object pointer
enum	enumerated type with a finite set of values (e.g. error codes)
record (struct)	a complex data type made of a sequence of other data types
SAFEARRAY(type)	array of (type) compatible with COM-technology. If an output parameter [out] is a SAFEARRAY the COM client has to destroy the SAFEARRAY when it is no longer needed (see the <code>SafeArrayDestroy Windows</code> function).

An asterisk * appearing after the data type refers to a 4 byte pointer to the data type.

Starting the COM server

COM server always has to be started by creating an *AxisVMApplication* object. As the COM server starts before AxisVM the user has to wait until AxisVM is loaded. There are two ways to detect whether AxisVM has been loaded or not.

- (1) Periodically checking the **Loaded** (Boolean) property of the *AxisVMApplication* object
- (2) Handling the **Loaded** event of the *AxisVMApplicationEvents* object.

If AxisVM is already running when COM server is started, the server connects to the running application. So the *AxisVMApplication.Loaded* property has to be checked first even if the second method is used. If the server connects to a running application the *AxisVMApplicationEvents.Loaded* event will never be called.

Error handling

There are two ways of error handling

- (1) checking the function result (in most cases a zero or negative integer value means an error)
- (2) handling events

General error codes for all interfaces:

EGeneralError

```
enum EGeneralError {  
    errDatabaseNotReady = -101,    model database is not available (AxisVM has not been loaded)  
    errNotFound = -102,           search item cannot be found  
    errIndexOutOfBounds = -103,   list index out of bounds  
    errReadOnly = -104            attempt to change a read only property  
}
```

Interface-specific error codes appear in interface descriptions.

Classes

Notation:

- A • after a property name indicates that the property can be written (i.e. not a read-only property)
- Input parameter of a function: [in]
- Output parameter of a function: [out]
- Parameter types appear *before* parameter names.
- Function result type appears *before* the function name.
- If the result type is void it means that the function has no return value.
- 0x...: hexadecimal value

IMPORTANT NOTICE:

If an output parameter [out] is a SAFEARRAY the COM client has to destroy the SAFEARRAY when it is no longer needed (see the SafeArrayDestroy Windows function).

IAxisVMApplication

The main interface of the COM server

Enumerated types

- enum **EMessageDialogButton** { **mdbHelp** = 0x1, **mdbCancel** = 0x2, **mdbOK** = 0x4, **mdbNo** = 0x8, **mdbYes** = 0x10, **mdbAbort** = 0x20, **mdbIgnore** = 0x40, **mdbRetry** = 0x80, **mdbAll** = 0x100}
Dialog buttons
- enum **EMessageDialogType** { **mdtWarning** = 0, **mdtError** = 1, **mdtInformation** = 2, **mdtConfirmation** = 3, **mdtCustom** = 4, **mdtErrorConfirmation** = 5 }
Dialog types

Functions

[EMessageDialogButton](#)* **MessageDlg** ([in] BSTR **Title**, [in] BSTR **Message**, [in] [EMessageDialogType](#) **DlgType**, [in] unsigned long **Buttons**)

Title *title of the dialog*

Message *message text of the dialog*

DlgType *dialog type*

Buttons *this value can be calculated by adding the [EMessageDialogButton](#) values of the required buttons*

*Shows a message dialog in AxisVM.
The result is the value of the button the user clicked.*

Properties

- BOOL **AskCloseOnLastReleased** •
Determines whether the COM server shutdown displays a query to close the AxisVM application as well.
- [AxisVMCatalog](#)* **Catalog**
AxisVM catalog object for standard materials and cross-sections.
- long **LibraryMajorVersion**
Major version number of the COM-server.
- long **LibraryMinorVersion**
Minor version number of the COM-server.
- BOOL **Loaded**
Shows if AxisVM has been loaded or not.
- [AxisVMModels](#)* **Models**
An object containing the opened models. The number of models is limited to one.
- BSTR **Version**
AxisVM program version string. Not the same as the COM-server version.
- BOOL **Visible** •

Determines if the main form of the AxisVM application is visible or not. Hiding the main window can considerably speed up certain operations. If AxisVM has been launched by the COM-server Visible remains False by default.

IAxisVMCatalog

AxisVM catalog object for standard materials and cross-sections.

Enumerated types

```
enum ECrossSectionType = {
    cstCustom = 0x00,           custom shape
    cstRectangular = 0x01,     rectangular shape
    cstI = 0x02,               I shape
    cstDoubleI = 0x03,         double I shape
    cstWedgedI = 0x04,         wedged I shape
    cstAsymmetricI = 0x05,     asymmetric I shape
    cstPipe = 0x06,           pipe
    cstRegularPolygon = 0x07,  regular polygon
    cstBox = 0x08,            box shape
    cstDoubleBox = 0x09,       double I box shape
    cstU = 0x0A,             U shape
    cstDoubleUOpened = 0x0B,   opened double U shape
    cstDoubleUClosed = 0x0C,   closed double U shape
    cstL = 0x0D,             unequal angle
    cstDoubleL = 0x0E,        double angle
    cstDoubleLFlange = 0x0F,   double angles connected at their flanges
    cstT = 0x10,             T shape
    cstZ = 0x11,             Z shape
    cstC = 0x12,             C shape
    cstS = 0x13,             S shape
    cstJ = 0x14,             J shape
    cstCircle = 0x15 }
    Cross-section types
enum ENationalDesignCode { ndcOther = 0x0, ndcHungarian_MSZ = 0x1,
    ndcEuroCode = 0x2, ndcRomanian_STAS = 0x4, ndcDutch_NEN = 0x5,
    ndcGerman_DIN1045_1 = 0x6, ndcSwiss_SIA26x = 0x7, ndcItalian = 0x9,
    ndcEuroCode_Austrian = 0xA, ndcEuroCode_UK = 0xB }
    National design codes
```

Functions

```
long GetCrossSection ([in] ECrossSectionType CrossSectionType, [in] BSTR TableName,
    [in] BSTR Name, [out] AxisVMCrossSection* CrossSection)
    CrossSectionType cross-section type
    TableName name of the cross-section table
    Name name of the cross-section in the table
    CrossSection an instance of the cross-section object
    Reads a cross-section from the catalog. If successful, result is > 0. Possible error codes:
    errNotFound (the cross-section cannot be found in the table). (To add a cross-section
    to the model use the AddFromCatalog function of the IAxisVMCrossSections interface).
```

```
long GetCrossSectionNames ([in] ECrossSectionType CrossSectionType,
    [in] BSTR TableName, [out] SAFEARRAY(BSTR)* Names)
    CrossSectionType cross-section type
    TableName name of the cross-section table
```

Names list of cross-section names

*Gets a list of cross-section names of the given type from a cross-section table.
Returns the number of list items.*

long **GetCrossSectionTableNames** ([in] [ECrossSectionType](#) **CrossSectionType**,
[out] SAFEARRAY(BSTR)* **TableNames**)

CrossSectionType cross-section type

TableNames list of cross-section table names

*Gets a list of cross-section table names of the given type.
Returns the number of list items.*

long **GetMaterial** ([in] [ENationalDesignCode](#) **DesignCode**, [in] BSTR **Name**,
[out] [AxisVMMaterial](#)* **Material**)

DesignCode national design code of the material

Name material name

Material an instance of the material object

*Reads a material from the catalog. If successful, result is > 0. Possible error codes:
errNotFound (the material of the given design code cannot be found). (To add a
material to the model use the [AddFromCatalog](#) function of the [IAxisVMMaterials](#)
interface).*

long **GetMaterialNames** ([in] [ENationalDesignCode](#) **DesignCode**,
[out] SAFEARRAY(BSTR)* **Names**)

DesignCode national design code of the material

Names list of material names

*Gets a list of material names of the given design code.
Returns the number of list items.*

IAxisVMModels

Contains the opened AxisVM models. The number of opened models is limited to one.

Functions

long **New**

Creates a new model. Returns the index of the new model.

Properties

long **Count**

The number of opened models. It is always 1.

[AxisVMModel](#)* **Item** [long **Index**]

Model object by its index. Index must be 1.

IAxisVMMModel

The opened AxisVM Model

Enumerated types

enum **EDisplay** = {
 dWireframe = 0x0, *wireframe*
 dHidden = 0x1, *wireframe with hidden lines removed*
 dRendered = 0x2, *rendered view*
 dTextured = 0x3 } *rendered view with textures*
Display mode of the model.

enum **EView** = {

vFront = 0x0, *front view (X-Z)*
vTop = 0x1, *top view (X-Y)*
vSide = 0x2, *side view (Y-Z)*
vPerspective = 0x3 } *perspective view*
View modes.

Functions

FitInView

Scales the model drawing so that it fits in the window.

BOOL **LoadFromFile** ([in] BSTR **FileName**)

FileName *file name*

Loads the model from a file. If successful, returns True, otherwise False.

BOOL **SaveToFile** ([in] BSTR **FileName**, [in] BOOL **SaveResults**)

FileName *file name*

SaveResults *save result file as well*

Saves the model as FileName.axs. If SaveResults = True, the result file is also saved as FileName.axe. If the operation was successful, returns True, otherwise False.

Properties

[AxisVMCalculation](#)* **Calculation**

Solver object of AxisVM.

[AxisVMCrossSections](#)* **CrossSections**

Cross-sections of the model.

[EDisplay](#) **Display** •

Current display mode of the model.

[AxisVMDomains](#)* **Domains**

Domains of the model.

[AxisVMLines](#)* **Lines**

Lines of the model.

[AxisVMLineSupports](#)* **LineSupports**

Line supports of the model.

[AxisVMLoadCases](#)* **LoadCases**

Load cases of the model.

[AxisVMLoadCombinations](#)* **LoadCombinations**

Load combinations of the model.

[AxisVMLoadGroups](#)* **LoadGroups**

Load groups of the model.

[AxisVMLoads](#)* **Loads**

Loads of the model.

[AxisVMMaterials](#)* **Materials**

Materials of the model.

BOOL **NeedsSaving**

True, if the model has been modified so it needs saving.

[AxisVMNodalSupports](#)* **NodalSupports**

Nodal supports of the model.

[AxisVMNodes](#)* **Nodes**

Nodes of the model.

[AxisVMReferences](#)* **References**

References of the model.

[AxisVMSettings](#)* **Settings**

Model settings.

[AxisVMSurfaces](#)* **Surfaces**
Surface elements of the model.

[AxisVMSurfaceSupports](#)* **SurfaceSupports**
Surface supports of the model.

[EView](#) **View** •
Current view of the model.

IAxisVMCalculation

Solver object of AxisVM.

Enumerated types

enum **EAxis** = { **Ax** = 0x0, **Ay** = 0x1, **Az** = 0x2 }
In case of displacement controlled nonlinear analysis it specifies the displacement component (X, Y, Z)

enum **ECalculationUserInteraction** = {
cuiUserInteraction = 0x0, *user has to interact with the program to answer questions arising during the analysis (e.g. domains are not meshed)*
cuiNoUserInteractionWithAutoCorrect = 0x1, *AxisVM tries to correct problems automatically during the analysis*
cuiNoUserInteractionWithoutAutoCorrect = 0x2, *AxisVM stops if a problem is detected during the analysis*
Sets the way of interaction between the user and the solver.

enum **EMassControl** = {
mcConvertLoadToMasses = 0x0, *converting loads to masses*
mcMassesOnly = 0x1 } *only masses are taken into account*
Setting used in vibration analysis.

enum **EReinforcementCalculation** = { **rcActual** = 0x0, **rcCalculated** = 0x1 }
Determines that if the analysis takes into account the reinforcement it uses the actual of the calculated one.

enum **ESolutionControl** = {
scForce = 0x0, *force control (equal load steps)*
scDisplacement = 0x1, *displacement control (equal displacement steps)*
scArcLength = 0x2, *arc length control*
scPushOver = 0x3 } *pushover analysis*
Way of solution control for nonlinear analysis.

Records / structures

RNonLinearAnalysis = (
 long **LoadCase** *load case to analyse*
 [ESolutionControl](#) **SolutionControl** *way of solution control*
 long **NodeId** *index of the node for displacement control*
 [EAxis](#) **Direction** *direction of displacement for displacement control*
 double **MaxDisplacement** *maximum displacement for displacement control [m]*
 long **Increments** *number of increments*
 double **DisplacementConvergenceValue** *convergence criterion for displacement*
 double **ForceConvergenceValue** *convergence criterion for force*
 double **WorkConvergenceValue** *convergence criterion for work*
 BOOL **EnableDisplacementConvergence** *DisplacementConvergenceValue enabled*
 BOOL **EnableForceConvergence** *ForceConvergenceValue enabled*
 BOOL **EnableWorkConvergence** *WorkConvergenceValue enabled*
 BOOL **GeometricNonlinearity** *taking into account the geometric nonlinearity*
 BOOL **ReinforcementCalculation** *taking into account the reinforcement*
 BOOL **StoreLastIncrementOnly** *only the last increment is stored*
 [EReinforcementCalculation](#) **ReinforcementCalculation** *use actual or calculated reinforcement*
) *if ReinforcementCalculation = True*

RVibration = (
)

long	LoadCase	<i>load case to analyse</i>
long	Iterations	<i>maximum number of iterations</i>
long	ModeShapes	<i>number of vibration shapes to determine</i>
double	EigenValueConvergence	<i>eigenvalue convergence criterion</i>
double	EigenVectorConvergence	<i>eigenvector convergence criterion</i>
BOOL	ConvertLoadToMasses	<i>convert load to masses</i>
BOOL	ConcentratedMasses	<i>use concentrated masses only</i>
BOOL	ConvertConcentratedMassesToLoads	<i>convert concentrated masses to loads</i>
EMassControl	MassControl	<i>way of mass control</i>
BOOL	ElementMasses	<i>taking into account the element masses</i>
BOOL	MassComponentX	<i>taking into account the mX mass component</i>
BOOL	MassComponentY	<i>taking into account the mY mass component</i>
BOOL	MassComponentZ	<i>taking into account the mZ mass component</i>
BOOL	ConvertSlabsToDiaphragms	<i>temporary conversion of slabs to diaphragms</i>
)	
	RBuckling = (
long	LoadCase	<i>load case to analyse</i>
long	Iterations	<i>maximum number of iterations</i>
long	ModeShapes	<i>number of buckling shapes to determine</i>
double	EigenValueConvergence	<i>eigenvalue convergence criterion</i>
double	EigenVectorConvergence	<i>eigenvector convergence criterion</i>
)	

Functions

BOOL	LinearAnalysis ([in] ECalculationUserInteraction UserInteraction)	<i>Linear analysis of the model. Returns True if the analysis was completed.</i>
BOOL	NonLinearAnalysis ([in] RNonLinearAnalysis AnalysisParameters , [in] ECalculationUserInteraction UserInteraction)	<i>Nonlinear analysis of the model. Returns True if the analysis was completed.</i>
BOOL	Vibration ([in] RVibration AnalysisParameters , [in] ECalculationUserInteraction UserInteraction)	<i>First order vibration analysis of the model. Returns True if the analysis was completed.</i>
BOOL	NonLinearVibration ([in] RVibration AnalysisParameters , [in] ECalculationUserInteraction UserInteraction)	<i>Second order vibration analysis of the model. Returns True if the analysis was completed.</i>
BOOL	Buckling ([in] RBuckling AnalysisParameters)	<i>Buckling analysis of the model. Returns True if the analysis was completed.</i>

IAxisVMCrossSections

Cross-sections of the model.

Error codes

enum	ECrossSectionError = {	
	cseNotAllowedProcessForShape = -100001	<i>process information is not compatible with the shape</i>
	cseNotAllowedProcessForParameters = -100002	<i>process information is not compatible with the parameters</i>
	cseNonPositive_b = -100003	<i>b ≤ 0</i>
	cseNonPositive_h = -100004	<i>h ≤ 0</i>
	cseNonPositive_tw = -100005	<i>tw ≤ 0</i>
	cseNonPositive_tf = -100006	<i>tf ≤ 0</i>
	cseNonPositive_d = -100007	<i>d ≤ 0</i>
	cseNonPositive_v = -100008	<i>v ≤ 0</i>
	cseNonPositive_b1 = -100009	<i>b1 ≤ 0</i>
	cseNonPositive_h1 = -100010	<i>h1 ≤ 0</i>
	cseNonPositive_b2 = -100011	<i>b2 ≤ 0</i>
	cseNonPositive_h2 = -100012	<i>h2 ≤ 0</i>
	cseNonPositive_tw1 = -100013	<i>tw1 ≤ 0</i>
	cseNonPositive_tf1 = -100014	<i>tf1 ≤ 0</i>

cseNonPositive_tw2 = -100015	$tw2 \leq 0$
cseNonPositive_tf2 = -100016	$tf2 \leq 0$
cseNonPositive_R = -100017	$R \leq 0$
cseNegative_r1 = -100018	$r1 < 0$
cseNegative_r2 = -100019	$r2 < 0$
cseNegative_r3 = -100020	$r3 < 0$
cseNegative_e = -100021	$e < 0$
cseNegative_a = -100022	$a < 0$
cseTooHigh_h = -100023	h too large
cseTooHigh_tw = -100024	tw too large
cseTooHigh_tf = -100025	tf too large
cseTooHigh_r1 = -100026	$r1$ too large
cseTooHigh_r2 = -100027	$r2$ too large
cseTooHigh_r3 = -100028	$r3$ too large
cseTooHigh_v = -100029	v too large
cseTooLow_e = -100030	e too small
cseTooHigh_tw1 = -100031	$tw1$ too large
cseTooHigh_tf1 = -100032	$tf1$ too large
cseTooHigh_tw2 = -100033	$tw2$ too large
cseTooHigh_tf2 = -100034	$tf2$ too large
cseTooLow_h = -100035	h too small
cseTooLow_r1 = -100036	$r1$ too small
cseTooLow_N = -100037	N too small
cseDifferentThicknesses = -100038	$v \neq t$
cseDifferentWidthAndHeight = -100039	$b \neq h$
cseIncompatibleWidthAndHeight = -100040	$b \neq 2h$
cseNonPositiveAx = -100041	$Ax \leq 0$
cseNegativeAy = -100042	$Ay < 0$
cseAylsHigherThanAx = -100043	$Ay > Ax$
cseNegativeAz = -100044	$Az < 0$
cseAzlsHigherThanAx = -100045	$Ay > Ax$
cseNonPositiveIx = -100046	$Ix \leq 0$
cseNonPositiveIy = -100047	$Iy \leq 0$
cseNonPositiveIz = -100048	$Iz \leq 0$
cseNonPositiveHy = -100049	$Hy \leq 0$
cseNonPositiveHz = -100050	$Hx \leq 0$
cseNegativeIw = -100051	$Iw < 0$
cseNegativeW1t = -100052	$W1t < 0$
cseNegativeW1b = -100053	$W1b < 0$
cseNegativeW2t = -100054	$W2t < 0$
cseNegativeW2b = -100055	$W2b < 0$
cseNegativeW1pl = -100056	$W1pl < 0$
cseNegativeW2pl = -100057	$W2pl < 0$
cseEmptyName = -100058	cross-section name is empty
cseNameAlreadyExists = -100059	cross-section name already exists
cseExtParams = -100060	error in extended parameters
cseException = -101000 }	other error

Cross-section error codes.

Enumerated types

```
enum ECrossSectionProcess = {
    cspOther = 0x0,          other
    cspRolled = 0x1,       rolled
    cspWelded = 0x2,       welded
    cspColdFormed = 0x3 } cold-formed
    How the cross-section was manufactured.

enum ECrossSectionDoubleUType = { duOpened = 0x0, duClosed = 0x1}
    Tells two types of double U-shapes apart (cstDoubleUOpened, cstDoubleUClosed).
```

Functions

There are two ways to create new cross-sections in the model. The **Add...** functions add a new cross-section, the **ReplaceWith...** functions replace an existing cross-section with another one.

If these functions return a value < 1 , it means that the result is one of the error codes of [EGeneralError](#) or [ECrossSectionError](#).

Successful **Add...** function calls return the index of the new cross-section within the IAxisVMCrossSections object.

Successful **ReplaceWith...** function calls return the index of the cross-section where the shape has been replaced. All geometry data is measured in meters.

long **AddAsymmetricI** ([in] BSTR Name, [in] double h, [in] double b1, [in] double tw, [in] double tf1, [in] double b2, [in] double tf2)

Name name of the new cross-section
h cross-section height
b1 cross-section upper flange width
tw cross-section web thickness
tf1 cross-section upper flange thickness
b2 cross-section lower flange width
tf2 cross-section lower flange thickness

Adds an asymmetric I-shape to the model.

long **AddBox** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section wall thickness at the web
tf cross-section wall thickness at the flange
R fillet radius

Adds a box shape to the model.

long **AddC** ([in] BSTR Name, [in] double h, [in] double b, [in] double e, [in] double tw, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
e leg height
tw wall thickness
R fillet radius
Process the manufacturing process

Adds a C-shape to the model.

long **AddCircle** ([in] BSTR Name, [in] double d)

Name name of the new cross-section
d circle diameter

Adds a circle shape to the model.

long **AddCustom** ([in] BSTR Name, [in] [AxisVMPolygon2dList](#)* ShapePolygonList, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
ShapePolygonList polygon list describing the shape
Process the manufacturing process

Adds a custom cross-section to the model with the given geometry.

long **AddDoubleI** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] double a, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
a distance of the I-shapes (zero for touching shapes)
Process the manufacturing process

Adds a double I-shape to the model.

long **AddDoubleIBox** ([in] BSTR Name, [in] double h, [in] double b, [in] double b1, [in] double tw, [in] double tf)

Name name of the new cross-section
h cross-section height
b cross-section width
b1 width of the box opening
tw cross-section web thickness
tf cross-section flange thickness

Adds a double I box shape to the model.

long **AddDoubleL** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double r1, [in] double r2, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
r1 fillet radius at the web/flange connection
r2 fillet radius at the ends
a the distance of angle shapes (zero for touching shapes)
Process the manufacturing process

Adds a double angle shape to the model.

long **AddDoubleLFlange** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double r1, [in] double r2, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
r1 fillet radius at the web/flange connection
r2 fillet radius at the ends
a the distance of angle shapes (zero for touching shapes)
Process the manufacturing process

Adds a double angle shape connected at the flanges to the model.

long **AddDoubleU** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] double a, [in] [ECrossSectionDoubleUType](#) OpenedClosed, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
a the distance of U-shapes (zero for touching shapes)
OpenedClosed opened or closed shape
(*cstDoubleUOpened* or *cstDoubleUClosed*)
Process the manufacturing process

Adds a double U-shape to the model.

long **AddFromCatalog** ([in] [ECrossSectionType](#) CrossSectionType, [in] BSTR CrossSectionName)

CrossSectionType cross-section type
CrossSectionName cross-section name

*Adds a cross-section from the catalog to the model.
Returns the same value as the other Add... functions.*

long **AddI** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process

Adds an I-shape to the model.

long **AddL** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double r1, [in] double r2, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
r1 fillet radius at the web/flange connection
r2 fillet radius at the ends
Process the manufacturing process

Adds an unequal angle shape to the model.

long **AddPipe** ([in] BSTR Name, [in] double d, [in] double v)

Name name of the new cross-section
d external diameter of the pipe
v pipe wall thickness

Adds a pipe to the model.

long **AddRectangular** ([in] BSTR Name, [in] double b, [in] double v)

Name name of the new cross-section
b cross-section width
v cross-section height

Adds a rectangular shape to the model.

long **AddRegularPolygon** ([in] BSTR Name, [in] long N, [in] double Rshape, [in] double v, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
N number of polygon sides
Rshape diameter of the circumcircle
v cross-section wall thickness
R fillet radius
Process the manufacturing process

Adds a regular polygon shape to the model.

long **AddReverseT** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process

Adds a reversed T-shape to the model.

long **AddT** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process

Adds a T-shape to the model.

long **AddU** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process

Adds a U-shape to the model.

long **AddWedgedI** ([in] BSTR Name, [in] double h1, [in] double b1, [in] double tw1, [in] double tf1, [in] double h2, [in] double b2, [in] double tw2, [in] double tf2, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h1 upper I-shape height
b1 upper I-shape width
tw1 upper I-shape web thickness
tf1 upper I-shape flange thickness
h2 lower I-shape height
b2 lower I-shape width
tw2 lower I-shape web thickness
tf2 lower I-shape flange thickness
R fillet radius
Process the manufacturing process

Adds a wedged I-shape to the model.

long **AddZ** ([in] BSTR Name, [in] double h, [in] double b, [in] double tw, [in] double tf, [in] double R, [in] [ECrossSectionProcess](#) Process)

Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process

Adds a Z-shape to the model.

void **Clear**

Removes all cross-sections from the model.

long **Delete** ([in] long Index)

Index number of the cross-section, $1 \leq \text{Index} \leq \text{Count}$

Deletes the cross-section specified by Index. If successful returns the Index, in case of

error returns an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **IndexOf** ([in] BSTR Name)
Name name of the cross-section to look for
Finds the cross-section in the list by name. If the cross-section is found returns its Index (> 0). Otherwise returns an error code ([errDatabaseNotReady](#) or [errNotFound](#)).

long **Merge** ([in] long **IntolItem**, [in] long **FromItem**, [in] double **OffsetY**, [in] double **OffsetZ**, [in] double **Rotation**, [in] BOOL **MergeIntoNew**)
IntolItem index of the destination cross-section
FromItem index of the source cross-section
OffsetY Y coordinate of the centre of gravity of the source shape in the coordinate system of the destination shape
OffsetZ Z coordinate of the centre of gravity of the source shape in the coordinate system of the destination shape
Rotation angle of rotation of the source shape around its centre of gravity
MergeIntoNew If set to True, the function merges the two shapes into a new cross-section. If set to False, the merged cross-section will replace the destination cross-section.
Merges two cross-sections. If **MergeIntoNew** is True returns the index of the new cross-section. If **MergeIntoNew** is False, returns the destination index (**IntolItem**). In case of an error the result is the error code.

long **ReplaceWithAsymmetricI** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b1**, [in] double **tw**, [in] double **tf1**, [in] double **b2**, [in] double **tf2**)
Index cross-section to replace
Name name of the new cross-section
h cross-section height
b1 cross-section upper flange width
tw cross-section web thickness
tf1 cross-section upper flange thickness
b2 cross-section lower flange width
tf2 cross-section lower flange thickness
Replaces the specified cross-section with an asymmetric I-shape.

long **ReplaceWithBox** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**)
Index cross-section to replace
Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section wall thickness at the web
tf cross-section wall thickness at the flange
R fillet radius
Replaces the specified cross-section with a box shape.

long **ReplaceWithC** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **e**, [in] double **tw**, [in] double **R**, [in] ECrossSectionProcess **Process**)
Index cross-section to replace
Name name of the new cross-section
h cross-section height
b cross-section width
e leg height
tw wall thickness
R fillet radius
Process the manufacturing process

Replaces the specified cross-section with a C-shape.

long **ReplaceWithCircle** ([in] long **Index**, [in] BSTR **Name**, [in] double **d**)
Index *cross-section to replace*
Name *name of the new cross-section*
d *diameter of the circle*

Replaces the specified cross-section with a circle shape.

long **ReplaceWithCustom** ([in] long **Index**, [in] BSTR **Name**,
[in] AxisVMPolygon2dList* **ShapePolygonList**, [in] ECrossSectionProcess **Process**)
Index *cross-section to replace*
Name *name of the new cross-section*
ShapePolygonList *polygon list describing the shape*
Process *the manufacturing process*

Replaces the specified cross-section with a custom shape.

long **ReplaceWithDoubleI** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**,
[in] double **tw**, [in] double **tf**, [in] double **R**, [in] double **a**,
[in] ECrossSectionProcess **Process**)
Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
a *the distance of the I-shapes (zero for touching shapes)*
Process *the manufacturing process*

Replaces the specified cross-section with a double I-shape.

long **ReplaceWithDoubleIBox** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**,
[in] double **b1**, [in] double **tw**, [in] double **tf**)
Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
b1 *width of the box opening*
tw *cross-section web thickness*
tf *cross-section flange thickness*

Replaces the specified cross-section with a double I box shape.

long **ReplaceWithDoubleL** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**,
[in] double **tw**, [in] double **tf**, [in] double **r1**, [in] double **r2**,
[in] ECrossSectionProcess **Process**)
Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
r1 *fillet radius at the web/flange connection*
r2 *fillet radius at the ends*
a *the distance of the angles (zero for touching shapes)*
Process *the manufacturing process*

Replaces the specified cross-section with a double angle shape.

long **ReplaceWithDoubleFlange** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **r1**, [in] double **r2**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
r1 *fillet radius at the web/flange connection*
r2 *fillet radius at the ends*
a *the distance of the angles (zero for touching shapes)*
Process *the manufacturing process*

Replaces the specified cross-section with a double angle shape connected at the flanges.

long **ReplaceWithDoubleU** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] double **a**, [in] ECrossSectionDoubleUType **OpenedClosed**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
a *the distance of the U-shapes (zero for touching shapes)*
OpenedClosed *opened or closed shape*
(cstDoubleUOpened or cstDoubleUClosed)
Process *the manufacturing process*

Replaces the specified cross-section with a double U-shape.

long **ReplaceWithI** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with an I-shape.

long **ReplaceWithL** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **r1**, [in] double **r2**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
r1 *fillet radius at the web/flange connection*
r2 *fillet radius at the ends*
Process *the manufacturing process*

Replaces the specified cross-section with an unequal angle shape.

long **ReplaceWithPipe** ([in] long **Index**, [in] BSTR **Name**, [in] double **d**, [in] double **v**)

Index *cross-section to replace*
Name *name of the new cross-section*
d *external diameter of the pipe*
v *pipe wall thickness*

Replaces the specified cross-section with a pipe.

long **ReplaceWithRectangular** ([in] long **Index**, [in] BSTR **Name**, [in] double **b**, [in] double **v**)

Index *cross-section to replace*
Name *name of the new cross-section*
b *cross-section width*
v *cross-section height*

Replaces the specified cross-section with a rectangular shape.

long **ReplaceWithRegularPolygon** ([in] long **Index**, [in] BSTR **Name**, [in] long **N**, [in] double **Rshape**, [in] double **v**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
N *number of polygon sides*
Rshape *diameter of the circumcircle*
v *wall thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a regular polygon shape.

long **ReplaceWithReverseT** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a reverse T-shape.

long **ReplaceWithT** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a T-shape.

long **ReplaceWithU** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*

R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a U-shape.

long **ReplaceWithWedgedI** ([in] long **Index**, [in] BSTR **Name**, [in] double **h1**, [in] double **b1**, [in] double **tw1**, [in] double **tf1**, [in] double **h2**, [in] double **b2**, [in] double **tw2**, [in] double **tf2**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h1 *upper I-shape height*
b1 *upper I-shape width*
tw1 *upper I-shape web thickness*
tf1 *upper I-shape flange thickness*
h2 *lower I-shape height*
b2 *lower I-shape width*
tw2 *lower I-shape web thickness*
tf2 *lower I-shape flange thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a wedged I-shape.

long **ReplaceWithZ** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **b**, [in] double **tw**, [in] double **tf**, [in] double **R**, [in] ECrossSectionProcess **Process**)

Index *cross-section to replace*
Name *name of the new cross-section*
h *cross-section height*
b *cross-section width*
tw *cross-section web thickness*
tf *cross-section flange thickness*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a Z-shape.

Properties

long **Count**
The number of cross-sections in the model.

[AxisVMCrossSection](#)* **Item** [long **Index**]
A cross-section of the model by Index. $1 \leq \text{Index} \leq \text{Count}$.

IAxisVMCrossSection

A cross-section in the model or in the catalog.

Records / structures

RStressPoint = (
double **y** *y coordinate of the stress point*
double **z** *z coordinate of the stress point*
)

Functions

long **AddStressPoint** ([in] **RStressPoint** **Point**)
Creates and adds a new stress point. Maximum number of stress calculation points is 9. The function returns the index of the new stress point. If result < 1, it is an error code ([errIndexOutOfBounds](#), if more stress points cannot be added).

long **DeleteStressPoint** ([in] long **Index**)
Deletes the stress point specified by Index. $1 \leq \text{Index} \leq \text{StressPointCount}$.

If successful, returns a positive value. Otherwise returns an error code (e. g. [errIndexOutOfBounds](#)).

long **Move** ([in] double **dy**, [in] double **dz**)
dy shift in y direction
dz shift in z direction

Shifts the cross-section.

If successful, returns the index of the new cross-section in the list of cross-sections ([AxisVMCrossSections](#)), otherwise returns an error code ([errReadOnly](#), if the cross-section is from the catalog or [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **Rotate** ([in] double **oy**, [in] double **oz**, [in] double **alpha**)
oy y coordinate of the rotation center
oz z coordinate of the rotation center
alpha angle of rotation

Rotates the cross-section around a centerpoint with the specified angle. Positive angles are counter-clockwise. If successful, returns the cross-section index in the list ([AxisVMCrossSections](#)), otherwise returns an error code ([errReadOnly](#), if the cross-section is from the catalog or [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **Mirror** ([in] double **y1**, [in] double **z1**, [in] double **y2**, [in] double **z2**)
y1 y coordinate of the 1st point of the mirroring axis
z1 z coordinate of the 1st point of the mirroring axis
y2 y coordinate of the 2nd point of the mirroring axis
z2 z coordinate of the 2nd point of the mirroring axis

Mirrors the cross-section to an axis specified by its two points.

If successful, returns the cross-section index in the list ([AxisVMCrossSections](#)), otherwise returns an error code ([errReadOnly](#), if the cross-section is from the catalog or [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **VerifyProperties**

Verifies cross-section properties.

If no error is found returns the index of the cross-section in the list ([AxisVMCrossSections](#)), otherwise returns an error-code ([errReadOnly](#), if the cross-section is from the catalog or [errDatabaseNotReady](#), [errIndexOutOfBounds](#), [ECrossSectionError](#)).

Properties

double **a** • (for double shapes) distance of the two shapes [m]

double **Alfa** rotation angle of parametric shapes. Positive angles are counter-clockwise. [rad]

double **Ax** • cross-section area [m²]

double **Ay** • shear area associated with shear forces in local y direction [m²]

double **Az** • shear area associated with shear forces in local z direction [m²]

double **b** • cross-section width [m]

double **b1** • (for box shapes) width of the opening [m]

double **b2** • (for wedged I-shapes) width of the lower I-shape [m]

[ECrossSectionType](#) **CrossSectionType** cross-section type

double **Cy** • y coordinate of the shear center relative to the centre of gravity [m]

double **Cz** • z coordinate of the shear center relative to the centre of gravity [m]

double **h** • cross-section height [m]

double **h2** • (for wedged I-shapes) height of the lower I-shape [m]

double **Hy** • y size of the bounding box of the rectangle [m]

double **Hz** • z size of the bounding box of the rectangle [m]

double **InnerPerimeter** *inner perimeter (sum of opening perimeters) [m]*

double **I1** *principal inertia about 1st local axis [m⁴]*

double **I2** *principal inertia about 2nd local axis [m⁴]*

double **Ialfa** *angle between the local 1st axis and the local y axis [rad]*

double **Ix** • *torsional inertia [m⁴]*

double **Iy** • *flexural inertia about local y axis [m⁴]*

double **Iw** • *warping modulus [m⁶]*

double **Iyz** • *centrifugal inertia [m⁴]*

double **Iz** • *flexural inertia about local z axis [m⁴]*

BOOL **Mirrored** *mirrored cross-section*

long **N** • *(for regular polygon shapes) number of polygon sides*

BSTR **Name** • *cross-section name*

double **OuterPerimeter** *outer perimeter (boundary perimeter)*

[ECrossSectionProcess](#) **Process** • *the manufacturing process*

double **r1** • *1. fillet radius [m]*

double **r2** • *2. fillet radius [m]*

double **r3** • *3. fillet radius [m]*

[AxisVMPolygon2dList](#) **ShapePolygonList** *polygon list describing the shape*

[RStressPoint](#) **StressPoint** [long **Index**] • *stress point by index (0 < Index ≤ StressPointCount)*

long **StressPointCount** *number of stress calculation points*

double **t** • *cross-section flange thickness [m]*

double **t2** • *(for wedged I-shapes) flange thickness for the lower I-shape [m]*

double **v** • *cross-section web thickness [m]*

double **v2** • *(for wedged I-shapes) web thickness for the lower I-shape [m]*

double **W1b** • *bottom elastic cross-section modulus for the 1st axis [m³]*

double **W1pl** • *plastic cross-section modulus for the 1st axis [m³]*

double **W1t** • *top elastic cross-section modulus for the 1st axis [m³]*

double **W2b** • *bottom elastic cross-section modulus for the 2nd axis [m³]*

double **W2pl** • *plastic cross-section modulus for the 2nd axis [m³]*

double **W2t** • *top elastic cross-section modulus for the 2nd axis [m³]*

double **Yg** • *position of the center of gravity of the cross-section in local y direction relative to the lower-left corner of the bounding rectangle [m]*

double **Zg** • *position of the center of gravity of the cross-section in local z direction relative to the lower-left corner of the bounding rectangle [m]*

IAxisVMDomains

Domains of the model

Enumerated types

```
enum ELineNonlinearity = {
    InlTensionAndCompression = 0x0,    linear behaviour
    InlTensionOnly = 0x1,              tension only
    InlCompressionOnly = 0x2 }        compression only
    Types of nonlinear behaviour.

enum EMeshType = {
    mtAdaptive = 0x0,                  adaptive mesh
    mtUniform = 0x1 }                 unofrm mesh
    Types of finite element mesh.

enum ESurfaceCharacteristics = {
```

schLinear = 0x0, *linear behaviour*
schTensionOnly = 0x1, *tension only*
schCompressionOnly = 0x2, *compression only*
schBilinear = 0x3 } *bilinear behaviour*

Nonlinear surface characteristics (not used).

```
enum ESurfaceType = {
    stHole = 0x0,                    hole
    stMembraneStress = 0x1,        membrane (plane stress)
    stMembraneStrain = 0x2,        tárca (plane strain)
    stPlate = 0x3,                   plate
    stShell = 0x4 }                 shell
```

Types of surface elements.

Error codes

```
enum EDomainsError = {
    deEmptyContour = -100001,        line list is empty
    deMoreThanOneContourFound = -100002    more than one contour can be identified in the line list
    deEmptyHole = -100003,            line list is empty (when defining a hole)
    deMoreThanOneHoleFound = -100004 }    more than one hole contour can be identified in the line list
    Errors during domain or hole definition
```

Records / structures

```
RDomainMeshParameters = (
    double MeshSize                    average mesh size [m]
    EMeshType MeshType                type of the finite element mesh
    BOOL IsFitToPointLoad            fit mesh to point loads
    double FitToPointLoad            minimum load to fit the mesh to the load point [kN]
    BOOL IsFitToLineLoad             fit mesh to line loads
    double FitToLineLoad             minimum load to fit the mesh to the load line [kN/m]
    BOOL IsFitToSurfaceLoad         fit mesh to surface loads
    double FitToSurfaceLoad         minimum load to fit the mesh to the load polygon [kN/m²]
)

RElasticFoundationXYZ = (
    double x, y, z                    elastic foundation stiffness in x, y, z directions [kN/m/m²]
)

RLineNonLinearity RNonLinearityXYZ = (
    x, y, z                            nonlinear behaviour in x, y, z directions
)

RResistancesXYZ = (
    double x, y, z                    resistance in x, y, z directions [kN/m²]
)

RSurfaceAttr = (
    double Thickness                 domain thickness [m]
    ESurfaceType SurfaceType         domain type
    long RefZId                        reference index for the local z direction
    (0 < RefZId ≤ AxisVMReferences.Count) or 0, if the reference is automatic
    long RefXId                        reference index for the local x direction
    (0 < RefXId ≤ AxisVMReferences.Count) or 0, if the reference is automatic
    long MaterialId                   index of the domain material
    (0 < MaterialId ≤ AxisVMMaterials.Count)
    RElasticFoundationXYZ ElasticFoundation    elastic foundation of the domain
    RNonLinearityXYZ NonLinearity            nonlinear behaviour of the elastic foundation
    RResistancesXYZ Resistance             resistance values of the elastic foundation
    ESurfaceCharacteristics Characteristics    nonlinear behaviour of the domain (not used)
)
```

Functions

```
long Add ([in] SAFEARRAY(long) Linelds, [in] RSurfaceAttr SurfaceAttr)
    Linelds    Line indices defining the domain.
               Lines must be in the same plane.
    SurfaceAttr    domain properties
    Defines a new domain.
```

If succesful, returns the new domain index, otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [deEmptyContour](#) [Linelds array is empty], [deMoreThanOneContourFound](#) [multiple contours]).

-
- long **Delete** ([in] double **Index**)
Index domain to delete
Deletes a domain. If successful, returns the domain index ($1 \leq \text{Index} \leq \text{Count}$), otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).
-
- long **DeleteSelected**
Deletes selected domains. If successful, returns the number of deleted domains, otherwise returns the error code ([errDatabaseNotReady](#)).
-
- long **GenerateMeshOnSelectedDomains** ([in] [RDomainMeshParameters](#) **MeshParameters**, [out] SAFEARRAY(long)* **ErrorCodes**, [out] SAFEARRAY(long)* **ErrorPoints**, [out] SAFEARRAY(long)* **ErrorLines**)
MeshParameters parameters for mesh generation
ErrorCodes list of error codes
ErrorPoints list of nodes causing error ([IAxisVMNodes](#))
ErrorLines list of lines causing error ([IAxisVMLines](#))
Generates a mesh for the selected domains. If successful, returns the number of deleted domains, otherwise returns the error code ([errDatabaseNotReady](#) or other negative numbers [in this case see ErrorCodes, ErrorPoints, ErrorLines for more information]).
-

Properties

- double **Count** number of domains in the model
[AxisVMDomain](#)* **Item** [long **Index**] a domain by index
 BOOL **Selected** [long **Index**] • gets or sets the selection status of a domain
 long **SelCount** number of selected domains in the model

IAxisVMDomain

An AxisVM domain object.

Records / structures

```

RMatrix3x3 = (
double  e11, e12, e13
double  e21, e22, e23
double  e31, e32, e33
)
RPoint3d = (
double  x, y, z           x, y, z coordinates of a 3D point or components of a 3D vector [m]
)

```

Functions

- long **AddHole** ([in] SAFEARRAY(long) **Linelds**)
Linelds Line indices defining a hole.
 Lines must be in the same plane.
*Defines a hole in the domain.
 If successful, returns the number of holes in the domain, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [deEmptyHole](#) [Linelds array is empty], [deMoreThanOneHoleFound](#) [multiple hole contours]).*
-
- long **Modify** ([in] SAFEARRAY(long) **Linelds**, [in] [RSurfaceAttr](#) **SurfaceAttr**)
Linelds line indices defining the domain

SurfaceAttr domain properties

Modifies domain geometry and / or domain properties. If successful, returns 0, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [deEmptyContour](#) [Linelds array is empty], [deMoreThanOneContourFound](#) [multiple contours]).

long **GenerateMesh** ([in] [RDomainMeshParameters](#) **MeshParameters**,
[out] SAFEARRAY(long)* **ErrorCodes**, [out] SAFEARRAY(long)* **ErrorPoints**,
[out] SAFEARRAY(long)* **ErrorLines**)

MeshParameters parameters for mesh generation

ErrorCodes list of error codes

ErrorPoints list of nodes causing error ([IAxisVMNodes](#))

ErrorLines list of lines causing error ([IAxisVMLines](#))

Generates a mesh for the domain. If successful, returns the domain index, otherwise returns an error code ([errDatabaseNotReady](#) or other negative numbers [in this case see [ErrorCodes](#), [ErrorPoints](#), [ErrorLines](#) for more information]).

Properties

double	Area domain area [m^2]
SAFEARRAY(long)*	ContourLines line indices of the contour
long	HoleCount number of holes in the domain
SAFEARRAY(long)*	HoleLines [long HoleIndex] line indices of a hole contour by index
BOOL	MeshExists tells if the domain is meshed or not
RDomainMeshParameters	MeshParameters • mesh parameters for the domain if MeshExists is True
SAFEARRAY(long)*	MeshSurfaceIds • an array of mesh surface element indices according to IAxisVMSurfaces
RPoint3d	NormalVector normal vector of the domain
RSurfaceAttr	SurfaceAttr • domain properties
RMatrix3x3	TrMatrix transformation matrix of the domain
double	Volume total volume of the domain [m^3]
double	Weight total mass of the domain [kg]

IAxisVMLines

Lines of the model.

Enumerated types

enum **ELineGeomType** = {
IgtStraightLine = 0x0, straight line
IgtCircleArc = 0x1 } circle or arc
Line geometry.

Records / structures

RCircleArcGeomData = (
[RPoint3d](#) **Center** arc centerpoint
[RPoint3d](#) **NormalVector** arc plane normal
double **Alpha** signed angle between the two endpoints [rad]. Positive angle is counter-clockwise if
seen from a direction opposite to the plane normal.
)
RLineGeomData = (
[RCircleArcGeomData](#) **CircleArc** arc geometry
[REllipseArcGeomData](#) **EllipseArc** ellipse arc geometry (**not used**)
)
RPoint3d = (
double **x, y, z** x, y, z coordinates of a 3D point or components of a 3D vector [m])

Functions

long **Add** ([in] long **StartNode**, [in] long **EndNode**, [in] [ELineGeomType](#) **GeomType**, [in] [RLineGeomData](#) **GeomData**,)

StartNode index of the startpoint
($0 < \text{StartNode} \leq \text{AxisVMNodes.Count}$)
EndNode index of the endpoint
($0 < \text{EndNode} \leq \text{AxisVMNodes.Count}$)
GeomType line geometry
GeomData geometry data

Adds a straight line or arc between two existing nodes.
If successful, returns the line index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **AddWithXYZ** ([in] [RPoint3d](#) **StartNode**, [in] [RPoint3d](#) **EndNode**, [in] [ELineGeomType](#) **GeomType**, [in] [RLineGeomData](#) **GeomData**,)

StartNode startpont coordinates
EndNode endpont coordinates
GeomType line geometry
GeomData geometry data

Adds a straight line or arc bewteen two points given by coordinates. Also creates two nodes at the endpoints with a nodal degree of freedom [dofFree](#).
If successful, returns the line index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **AddWithXYZDOF** ([in] [RPoint3d](#) **StartNode**, [in] [RPoint3d](#) **EndNode**, [in] long **StartDOF**, [in] long **EndDOF**, [in] [ELineGeomType](#) **GeomType**, [in] [RLineGeomData](#) **GeomData**,)

StartNode startpont coordinates
EndNode endpont coordinates
StartDOF nodal degrees of freedom for the start node
EndDOF nodal degrees of freedom for the end node
GeomType line geometry
GeomData geometry data

Adds a straight line or arc bewteen two points given by coordinates. Also creates two nodes at the endpoints. Nodal degrees of freedom will be **StartDOF** and **EndDOF**.
If successful, returns the line index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **Clear**

Removes all lines. It returns the number of lines removed. If it returns a negative number, that is an error code ([errDatabaseNotReady](#)).

long **DefineSelectedLinesAsRigidBody**

Redefines the rigid bodies of the model based on selected lines. If there were any rigid bodies in the model they are deleted if not selected. If successful, returns the number of rigid bodies in the model, otherwise returns the error code ([errDatabaseNotReady](#), [leNoLinesAreSelected](#) [see [IAxisVMLine](#)]).

long **Delete** ([in] double **Index**)

Index index of the line to delete

Deletes a line. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns the **Index**, otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **DeleteSelected**

Deletes the selected lines. If successful, returns the number of deleted lines, otherwise

returns the error code. ([errDatabaseNotReady](#)).

long **IndexOf** ([in] long **StartNode**, [in] long **EndNode**)

StartNode start node index of the line
($0 < \text{StartNode} \leq \text{AxisVMNodes.Count}$)

EndNode end node index of the line
($0 < \text{EndNode} \leq \text{AxisVMNodes.Count}$)

Retrieves the index of the line by its end nodes.
If successful, returns the index of the line, otherwise returns the error code ([errDatabaseNotReady](#), [errNotFound](#)).

long **IndexOf2** ([in] long **StartNode**, [in] long **EndNode**, [in] long **FirstLine**)

StartNode start node index of the line
($0 < \text{StartNode} \leq \text{AxisVMNodes.Count}$)

EndNode end node index of the line
($0 < \text{EndNode} \leq \text{AxisVMNodes.Count}$)

FirstLine search begins at this index

Retrieves the index of the line by its end nodes. Search begins at the specified index.
Call `IndexOf2` to find all lines between two nodes (e.g. different arcs). If successful, returns the index of the line, otherwise returns the error code ([errDatabaseNotReady](#), [errNotFound](#)).

long **SelectAll** ([in] BOOL **Select**)

Select selection state

If `Select` is `True`, selects all lines.

If `Select` is `False`, deselects all lines.

If successful, returns the number of selected lines, otherwise returns the error code ([errDatabaseNotReady](#)).

Properties

long **Count** number of lines in the model

[AxisVMLine](#)* **Item** [long **Index**] line object by index

long **RigidBodyCount** number of lines defined as rigid bodies

BOOL **Selected** [long **Index**] • gets or sets the selection status of a line

long **SelCount** number of selected lines in the model

IAxisVMLine

AxisVM line object.

Enumerated types

enum **EGapType** = {

agtActiveInTension = 0x0, gap active in tension only

agtActiveInCompression = 0x1 } gap active in compression only

Gap element type.

enum **ELineNonlinearity** = {

InlTensionAndCompression = 0x0, linear behaviour

InlTensionOnly = 0x1, tension only

InlCompressionOnly = 0x2 } compression only

Types of nonlinear behaviour.

enum **ELineType** = {

ItTruss = 0x0, truss

ItBeam = 0x1, beam

ItRib = 0x2, rib

ItSpring = 0x3, spring

```

ltGap = 0x4,           gap element
ltEdge = 0x5,         surface edge
ltHole = 0x6,         hole edge
ltSimpleLine = 0x7,   line without element properites
ltNNLink = 0x8,       node-to-node link element
ltLLLink = 0x9 }     line-to-line link element
Line type.

enum EReleaseType = {
    rtRigid = 0x0,       rigid
    rtHinged = 0x1,     hinged
    rtSemiRigid = 0x2,  semi-rigid
    rtPlastic = 0x3 }  plastic
Type of end release.

enum ESpringDirection = {
    sdGlobal = 0x0,     global
    sdGeometry = 0x1,   by geometry
    sdPointReference = 0x2, by reference point
    sdVectorReference = 0x3, by reference vector
    sdElementRelative = 0x4, by element
    sdNodeRelative = 0x5 } by node
Coordinate system for spring stiffness components.

```

Error codes

```

enum ELineError
leNodeIndexOutOfBounds = -100001  node index is out of bounds
leReferenceIndexOutOfBounds = -100002 reference index is out of bounds
leReadOnlyPropertyForThisLineType = -100003 the property is read-only for this line type
lePropertyNotValidForThisLineType = -100004 the property is not compatible with this line type
leMaterialIndexOutOfBounds = -100005 material index is out of bounds
leCrossSectionIndexOutOfBounds = -100006 cross-section index is out of bounds
leNoLinesAreSelected = -100007 no lines are selected

```

Records / structures

```

RCircleArcGeomData = (
    RPoint3d Center           arc centerpoint
    RPoint3d NormalVector       arc plane normal
    double Alpha           signed angle between the two endpoints [rad]. Positive angle is counter-clockwise if
seen from a direction opposite to the plane normal.
)

RLineGeomData = (
    RCircleArcGeomData CircleArc       arc geometry
    REllipseArcGeomData EllipseArc       ellipse arc geometry (not used)
)

double RPoint3d = (
    x, y, z           x, y, z coordinates of a 3D point or components of a 3D vector [m]
)

EReleaseType RRelease = (
    Release           release type (rigid / hinged / semi-rigid / plastic)
    double Init       (if Release = rtSemiRigid) initial rotational stiffness of the connection [kNm/rad]
    double Limit     (if Release = rtSemiRigid) moment resistance of the connection [kNm]
)

RReleases = (
    RRelease x           release in x direction (x.Release = rtRigid v. rtHinged)
    RRelease y           release in y direction (y.Release = rtRigid v. rtHinged)
    RRelease z           release in z direction (z.Release = rtRigid v. rtHinged)
    RRelease xx          release around the x axis (xx.Release = rtRigid v. rtHinged)
    RRelease yy          release around the y axis
    RRelease zz          release around the z axis
)

double RStiffnesses = (
    x, y, z           stiffness in x, y, z direction [kN/m]
    double xx, yy, zz rotational stiffness around x, y, z axes [kNm/rad]
)

```

Functions

-)
- long **DefineAsBeam** ([in] long **MaterialIndex**, [in] long **StartCrossSectionIndex**, [in] long **EndCrossSectionIndex**, [in] [RPoint3d](#) **StartEccentricity**, [in] [RPoint3d](#) **EndEccentricity**)
- | | |
|-------------------------------|---|
| MaterialIndex | index of the material
($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$) |
| StartCrossSectionIndex | index of the start cross-section
($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$) |
| EndCrossSectionIndex | index of the end cross-section
($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$) |
| StartEccentricity | eccentricity at the start node (not used) |
| EndEccentricity | eccentricity at the end node (not used) |
- Defines a line object as a beam. For beams with a constant cross-section $\text{StartCrossSectionIndex} = \text{EndCrossSectionIndex}$.
If successful, returns the line index according to [IAxisVMLines](#), otherwise returns the error code ([leCrossSectionIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).
-
- long **DefineAsGap** ([in] [EGapType](#) **GapType**, [in] double **ActiveStiffness**, [in] double **InactiveStiffness**, [in] double **InitialOpening**, [in] double **MinPenetration**, [in] double **MaxPenetration**, [in] double **AdjustmentRatio**)
- | | |
|---------------------------|--|
| GapType | gap type |
| Active Stiffness | gap stiffness when active [kN/m] |
| Inactive Stiffness | gap stiffness when inactive [kN/m] |
| Initial Opening | initial opening [m] |
| MinPenetration | minimum penetration [m] |
| MaxPenetration | maximum penetration [m] |
| AdjustmentRatio | adjustment ratio of the active stiffness |
- Defines a line object as a gap element.
If successful, returns the line index according to [IAxisVMLines](#), otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).
-
- long **DefineAsRib** ([in] long **MaterialIndex**, [in] long **StartCrossSectionIndex**, [in] long **EndCrossSectionIndex**, [in] [RPoint3d](#) **StartEccentricity**, [in] [RPoint3d](#) **EndEccentricity**)
- | | |
|-------------------------------|---|
| MaterialIndex | index of the material
($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$) |
| StartCrossSectionIndex | index of the start cross-section
($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$) |
| EndCrossSectionIndex | index of the end cross-section
($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$) |
| StartEccentricity | eccentricity at the start node (not used) |
| EndEccentricity | eccentricity at the end node (not used) |
- Defines a line object as a rib. For ribs with a constant cross-section $\text{StartCrossSectionIndex} = \text{EndCrossSectionIndex}$.
If successful, returns the line index according to [IAxisVMLines](#), otherwise returns the error code ([leCrossSectionIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).
-
- long **DefineAsSpring** ([in] [ESpringDirection](#) **SpringDirection**, [in] [RStiffnesses](#) **Stiffnesses**)
- | | |
|------------------------|---|
| SpringDirection | coordinate system of spring stiffnesses |
| Stiffnesses | stiffnesses |
- Defines a line object as a spring element.
If successful, returns the line index according to [IAxisVMLines](#), otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).
-

long **DefineAsTruss** ([in] long **MaterialIndex**, [in] long **CrossSectionIndex**, [in] **ELineNonLinearity** **TrussType**, [in] double **Resistance**)

MaterialIndex *index of the material*
(0 < MaterialIndex ≤ [AxisVMMaterials.Count](#))

CrossSectionIndex *index of the cross-section*
(0 < CrossSectionIndex ≤ [AxisVMCrossSections.Count](#))

TrussType *nonlinear behaviour*

Resistance *resistance [kN]*

Defines a line object as a truss element.

If successful, returns the line index according to [IAxisVMLines](#), otherwise returns the error code ([leCrossSectionIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

Properties

If the property (read or written) is not compatible with **LineType** or **GeomType**, an error event is created with the error code *lePropertyNotValidForThisLineType*. If **LineType** makes the property read-only and the client program tries to write it, an error event is created with the error code *leReadOnlyPropertyForThisLineType*. If the reference index is not valid, an error event is created with the error code *leReferenceIndexOutOfBounds*.

long **EndNode** • *index of the end node according to [IAxisVMNodes](#)*

[RReleases](#) **EndReleases** • *(only if LineType = ItBeam or ItRib) end releases*

[RLineGeomData](#) **GeomData** • *(if GeomType = lgtCircleArc) geometry data for the arc*

[ELineGeomType](#) **GeomType** • *line geometry (straight line or arc)*

long **Length** *length of the line [m]*

[ELineType](#) **LineType** • *line element type*

[ELineNonLinearity](#) **NonLinearity** • *nonlinear behaviour (can be written only if LineType = ItTruss)*

long **Reference** • *index of the reference (according to [IAxisVMReferences](#) szerint). If an automatic reference is set, Reference = 0.*

[RReleases](#) **StartReleases** • *(only if LineType = ItBeam or ItRib) end releases at the start node*

double **TrussResistance** • *(only if LineType = ItTruss) resistance of the truss [kN]*

IAxisVMLineSupports

Line supports of the model.

Enumerated types

enum **ELineNonlinearity** = {

InITensionAndCompression = 0x0, *linear behaviour*

InITensionOnly = 0x1, *tension only*

InICompressionOnly = 0x2 } *compression only*

Types of nonlinear behaviour.

Records / structures

[ELineNonlinearity](#) **RNonlinearity** = (
x, y, z, *nonlinear behaviour in x, y, z direction*
xx, yy, zz *and around the x, y, z axis*
)

[ELineNonlinearity](#) **RNonlinearityXYZ** = (
x, y, z *nonlinear behaviour in x, y, z direction*
)

double **RResistances** = (
x, y, z, *resistances in x, y, z direction [kN/m]*
xx, yy, zz *and around the x, y, z axis [kNm/m]*
)

double **RResistancesXYZ** = (
x, y, z *resistances in x, y, z direction [kN/m]*
)

```

RStiffnesses = (
double x, y, z      stiffnesses in x, y, z direction [kN/m/m]
double xx, yy, zz  rotational stiffnesses around the x, y, z axis [kN/rad/m]
)

RStiffnessesXYZ = (
double x, y, z      stiffnesses in x, y, z direction [kN/m/m]
)

```

Functions

```

long AddBeamElasticFoundation ([in] long BeamId,
[in] RStiffnessesXYZ StiffnessesXYZ, [in] RNonLinearityXYZ NonlinearityXYZ,
[in] RResistancesXYZ ResistancesXYZ)

```

BeamId *index of the beam*
($0 < \text{BeamId} \leq \text{AxisVMLines.Count}$)

StiffnessesXYZ *stiffnesses of the elastic foundation*

NonlinearityXYZ *nonlinear behaviour of the elastic foundation*

ResistancesXYZ *resistances for stiffness components*

Adds an elastic foundation to a beam. If successful, returns the support index, otherwise returns the error code ([leInvalidLineType](#), if the line is not a beam or [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

```

long AddEdgeGlobal ([in] RStiffnesses Stiffnesses, [in] RNonLinearity Nonlinearity,
[in] RResistances Resistances, [in] long EdgeId, [in] long SurfaceId1,
[in] long SurfaceId2, [in] long DomainId1, [in] long DomainId2)

```

Stiffnesses *stiffness components in the global system*

Nonlinearity *nonlinear behaviour of the support*

Resistances *resistances for stiffness components*

EdgeId *index of the edge*
($0 < \text{EdgeId} \leq \text{AxisVMLines.Count}$)

SurfaceId1 *first connecting surface*
($0 < \text{SurfaceId1} \leq \text{AxisVMSurfaces.Count}$)

SurfaceId2 *second connecting surface*
($0 < \text{SurfaceId2} \leq \text{AxisVMSurfaces.Count}$)

DomainId1 *first connecting domain*
($0 < \text{DomainId1} \leq \text{AxisVMDomains.Count}$)

DomainId2 *second connecting domain*
($0 < \text{DomainId2} \leq \text{AxisVMDomains.Count}$)

Adds an edge support in global system. Connecting surfaces and domains are stored so that if all of them is deleted the support is deleted as well. If successful, returns the support index, otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

```

long AddEdgeRelative ([in] RStiffnesses Stiffnesses, [in] RNonLinearity Nonlinearity,
[in] RResistances Resistances, [in] long EdgeId, [in] long SurfaceId1,
[in] long SurfaceId2, [in] long DomainId1, [in] long DomainId2)

```

Stiffnesses *stiffness components in the local system of the edge*

Nonlinearity *nonlinear behaviour of the support*

Resistances *resistances for stiffness components*

EdgeId *index of the edge*
($0 < \text{EdgeId} \leq \text{AxisVMLines.Count}$)

SurfaceId1 *first connecting surface*
($0 < \text{SurfaceId1} \leq \text{AxisVMSurfaces.Count}$)

SurfaceId2 *second connecting surface*
($0 < \text{SurfaceId2} \leq \text{AxisVMSurfaces.Count}$)

DomainId1 *first connecting domain*
($0 < \text{DomainId1} \leq \text{AxisVMDomains.Count}$)

DomainId2 *second connecting domain*
($0 < \text{DomainId2} \leq \text{AxisVMDomains.Count}$)

Adds an edge support in global system. Connecting surfaces and domains are stored so that if all of them is deleted the support is deleted as well. If only one connecting

surface or domain is specified (other surface/domain indices are zero) the local x direction is along the edge, the local y direction is in the plane of the surface and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the surface. If two indices are nonzero the local x direction is along the edge, the local z direction is on the bisector of the local z directions of the two, the local y direction is perpendicular to both. If successful, returns the support index, otherwise returns the error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **AddRibElasticFoundation** ([in] long **RibId**,
[in] [RStiffnessesXYZ](#) **StiffnessesXYZ**, [in] [RNonLinearityXYZ](#) **NonlinearityXYZ**,
[in] [RResistancesXYZ](#) **ResistancesXYZ**)

RibId index of the rib
($0 < \text{RibId} \leq \text{AxisVMLines.Count}$)

StiffnessesXYZ stiffnesses of the elastic foundation

NonlinearityXYZ nonlinear behaviour of the elastic foundation

ResistancesXYZ resistances for stiffness components

Adds an elastic foundation to a rib. If successful, returns the support index, otherwise returns the error code ([IInvalidLineType](#), if the line is not a beam or [errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

Properties

long **Count** number of line supports in the model
[AxisVMLineSupport*](#) **Item** [long **Index**] line support object by index
 BOOL **Selected** [long **Index**] • gets of sets the selection status of a line support
 long **SelCount** number of selected line supports

IAxisVMLineSupport

An AxisVM line support object.

Enumerated types

enum **ELineSupportType** = {
IstEdgeGlobal = 0x0, global edge support
IstEdgeRelative = 0x1, edge relative support
IstRibElasticFoundation = 0x2, elastic foundation of a rib
IstBeamElasticFoundation = 0x3 } elastic foundation of a beam
 Line support type

Properties

If the support is an *IstRibElasticFoundation* or an *IstBeamElasticFoundation*, reading a property valid only for *IstEdgeGlobal* or *IstEdgeRelative* supports returns 0 and an attempt to write creates an error event.
 If the support is an *IstEdgeGlobal* or an *IstEdgeRelative*, reading a property valid only for *IstRibElasticFoundation* or *IstBeamElasticFoundation* supports returns 0 and an attempt to write creates an error event.

long **DomainId1** (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*)
 first domain connecting to the edge
 long **DomainId2** (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*)
 second domain connecting to the edge
 long **EdgeId** (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*)
 edge index according to [IAxisVMLines](#)
 long **LineId** line index according to [IAxisVMLines](#)
[RNonlinearity](#) **Nonlinearity** • (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*)
 nonlinear behaviour
[RNonlinearityXYZ](#) **NonlinearityXYZ** • (*SupportType* = *IstBeamElasticFoundation* or
IstRibElasticFoundation) nonlinear behaviour of the elastic foundation
[RResistances](#) **Resistances** • (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*)

resistance components of the support

[RResistancesXYZ](#) **ResistancesXYZ** • (*SupportType* = *IstBeamElasticFoundation* or *IstRibElasticFoundation*) resistance components of the elastic foundation

[RStiffnesses](#) **Stiffnesses** • (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*) support stiffnesses

[RStiffnessesXYZ](#) **StiffnessesXYZ** • (*SupportType* = *IstBeamElasticFoundation* or *IstRibElasticFoundation*) elastic foundation stiffnesses

[ELineSupportType](#) **SupportType** line support type

long **Surfaceld1** (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*) first surface connecting to the edge

long **Surfaceld2** (*SupportType* = *IstEdgeGlobal* or *IstEdgeRelative*) first surface connecting to the edge

IAxisVMLoadCases

Load cases in the model.

Enumerated types

```
enum ELoadCaseType = {
    lctStandard = 0x0,           standard load case
    lctInfluenceLine = 0x1,      influence line load case
    lctSeismic = 0x2,           seismic load case
    lctVibration = 0x3,         vibration load case
    lctPreStress = 0x4,         prestress load case
    lctMoving = 0x5 }          moving load case
    Load case type.

enum EModalCombType = {
    mctAuto = 0x0,              automatic
    mctSRSS = 0x1,             Square Root of Sum of Squares
    mctCQC = 0x2 }            Complete Quadratic Combination
    Combination type for modal responses in one direction

enum ESeismicCombType = {
    sctQuadratic = 0x0,        quadratic mean
    sctMax = 0x1,              combination with 30%
    sctAuto = 0x2 }           automatic
    Combination type for spatial components.

enum EVibrationType = {
    vtFirstOrder = 0x0,       first order vibration
    vtSecondOrder = 0x1 }     second order vibration
    Vibration result type.
```

Error codes

```
enum ELoadCasesError = {
    IcePropertyNotValidForThisType = -100001 } property is not compatible with the load case type
```

Records / structures

Fields with (MSz) are valid only if design code is Hungarian. Fields with (STAS) are valid only if design code is Romanian. Fields with (*) are valid only for non-Hungarian design codes.

```
RSeismicParams = (
    EVibrationType VibrType      vibration result type
    double kg                    (MSz)  $k_g$  seismic constant
    double ks                    (MSz)  $k_s$  importance factor of the building
    double kt                    (MSz)  $k_t$  soil factor
    double psi                   (MSz)  $\Psi$  damping factor
    ESeismicCombType SeismicCombType (*) combination type for spatial components
```

double	qd	(* if non-STAS) q_d behaviour factor for displacements
double	ksiV	(*) ξ damping factor
EModalCombType	ModalCombType	(*) combination type for modal responses in one direction
BOOL	Torsion	(*) tells if torsion effects are taken into account
double	ExcCoeff	(*) eccentricity coefficient
double	C	(STAS) C
double	nu	(STAS) ν

Functions

long **Add** ([in] BSTR Name, [in] [ELoadCaseType](#) LoadCaseType)

Name name of the load case

LoadCaseType load case type

Adds a new load case to the model. If successful, returns the load case index, otherwise returns the error code. ([errDatabaseNotReady](#)).

long **Delete** ([in] long Index)

Index index of the load case to delete ($0 < \text{Index} \leq \text{Count}$)

Deletes a load case by index.

If successful, returns Index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Properties

long **Count** number of load cases in the model. Negative number is an error code ([errDatabaseNotReady](#)).

long **GroupId** [long Index] • load group index of a load case by index. See [IAxisVMLoadGroups](#). Negative number is an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

[ELoadCaseType](#) **LoadCaseType** [long Index] load case type by index

long **LoadCount** [long Index] number of loads in a load case by index. Negative number is an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

BSTR **Name** [long Index] • load case name by index

[RSeismicParams](#) **SeismicParams** [long Index] (only if LoadCaseType is *IctSeismic* or *IctVibration*) seismic parameters of the load case according to the design code. For other types of load cases all fields are set to zero and an error event is created with the error code [IcePropertyNotValidForThisLoadType](#).

IAxisVMLoadCombinations

Load combinations of the model.

Enumerated types

```
enum ECombinationType = {
    ctManual = 0x0,           user combination
    ctULS = 0x1,             ULS (Ultimate Limit State) combination
    ctSLS = 0x2 }           SLS (Service Limit State) combination
Load combination type.
```

Error codes

```
enum ELoadCombinationsError = {
    IceDifferentFactorsAndIDsCount = -100001 } the number of load cases and factors are different
```

Functions

long **Add** ([in] BSTR Name, [in] [ECombinationType](#) CombinationType, [in] SAFEARRAY(double)* Factors, [in] SAFEARRAY(long)* LoadCaseIds)

Name name of the load combination

CombinationType *combination type*
Factors *combination factors*
LoadCaseIds *combination load case indices according to [IAxisVMLoadCases](#)*

*Adds a new load combination to the model.
If successful, returns the index of the new combination, otherwise returns the error code ([leDifferentFactorsandIDsCount](#), [errDatabaseNotReady](#)).*

long **Delete** ([in] long **Index**)
Index *index of the load combination to delete ($0 < Index \leq Count$)*
*Deletes a load combination.
If successful, returns Index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).*

long **GetCombination** ([in] long **Index**, [out] SAFEARRAY(double)* **Factors**, [out] SAFEARRAY(long)* **LoadCaseIds**)
Index *index of the load combination ($0 < Index \leq Count$)*
Factors *load combination factors*
LoadCaseIds *combination load case indices according to [IAxisVMLoadCases](#)*
*Retrieves a load combination.
If successful, returns Index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).*

long **SetCombination** ([in] long **Index**, [in] SAFEARRAY(double)* **Factors**, [in] SAFEARRAY(long)* **LoadCaseIds**)
Index *index of the load combination ($0 < Index \leq Count$)*
Factors *load combination factors*
LoadCaseIds *combination load case indices according to [IAxisVMLoadCases](#)*
*Modifies an existing load combination.
If successful, returns Index, otherwise returns the error code ([leDifferentFactorsandIDsCount](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).*

Properties

[ECombinationType](#) **CombinationType** [long **Index**] *type of a combination*
long **Count** *number of load combinations in the model
Negative number is an error code ([errDatabaseNotReady](#)).*
BSTR **Name** [long **Index**] • *name of a combination*

IAxisVMLoadGroups

Load groups in the model.

Enumerated types

enum **EGroupCombinationType** = {
gctOld = 0x0, **(not used)**
gctExclusive = 0x1, *(for permanent and prestress load groups): When determining critical combination only the most unfavourable load case will be taken into account from the load group with its upper or lower safety factor.
(for incidental load groups): only one load case of the group can be included into the critical combination.*
gctAdditive = 0x2} *(for permanent and prestress load groups): all load cases will be taken into account simultaneously when determining the critical combination.*

(for incidental load groups): multiple load cases can be included into the critical combination.

Behaviour of load cases of groups when determining the critical combination.

```
enum ELoadGroupType = {  
    lgtPermanent = 0x0,    permanent load group  
    lgtIncidental = 0x1,   incidental load group  
    lgtExceptional = 0x2,  exceptional load group  
    lgtSeismic = 0x3,     seismic load group  
    lgtPrestress = 0x4,   prestress load group  
    lgtMoving = 0x5 }  
Load group types.
```

Error codes

```
enum ELoadGroupsError = {  
    lgePropertyNotValidForThisType = -100001 }    property is not compatible with the load group type
```

Functions

```
long Add ([in] BSTR Name, [in] ELoadGroupType LoadGroupType,  
          [in] BOOL SimultExc, [in] EGroupCombinationType CombinationType)  
    Name    name of the load group  
    LoadGroupType load group type  
    SimultExc (if LoadGroupType = lgtIncidental) sets if load cases of  
              the group can be simultaneous with load cases of  
              exceptional groups  
    CombinationType Behaviour of load cases of groups when determining the  
                    critical combination.
```

Adds a new load group to the model.

If successful, returns the index of the new group, otherwise returns the error code ([errDatabaseNotReady](#)).

```
long Delete ([in] long Index)  
    Index index of the load group to delete (0 < Index ≤ Count)
```

Deletes a load group.

If successful, returns **Index**, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Properties

```
long Count number of load groups in the model.  
Negative number is an error code (errDatabaseNotReady).
```

```
AxisVMLoadGroup* Item [long Index] • load group object by index
```

IAxisVMLoadGroup

An AxisVM load group object.

Error codes

```
enum ELoadGroupsError = {  
    lgePropertyNotValidForThisType = -100001 }    property is not compatible with the current design code
```

Properties

Only properties compatible with the current design code can be read or written. Incompatibility creates an error event with the error code [lgePropertyNotValidForThisType](#). Properties with (EC) are valid in EC, SIA, DIN, STAS, I design codes.

```
EGroupCombinationType CombinationType • behaviour of load cases of groups when determining the critical
```


ItBeamFault = 0x05,	<i>"fault in length" load on beams</i>
ItBeamSelfWeight = 0x07,	<i>self weight load on beams</i>
ItTrussThermal = 0x08,	<i>thermal load on trusses</i>
ItTrussStress = 0x09,	<i>tension/compression load on trusses</i>
ItTrussFault = 0x0A,	<i>"fault in length" load on trusses</i>
ItTrussSelfWeight = 0x0B,	<i>self weight load on trusses</i>
ItSurfaceSelfWeight = 0x0C,	<i>self weight load on surface elements</i>
ItSurfaceDistributed = 0x0D,	<i>distributed load on surface elements</i>
ItSurfaceEdge = 0x0E,	<i>edge load on surface elements</i>
ItSurfaceThermal = 0x0F,	<i>thermal load on surface elements</i>
ItSurfaceStress = 0x10,	(not used)
ItBeamInfluence = 0x11,	<i>influence line load on beams</i>
ItDomainSelfWeight = 0x12,	<i>self weight load on domains</i>
ItDomainDistributed = 0x13,	<i>distributed load on domains</i>
ItDomainEdge = 0x14,	<i>edge load on domains</i>
ItDomainThermal = 0x15,	<i>thermal load on domains</i>
ItDomainStress = 0x16,	(not used)
ItRibThermal = 0x17,	<i>thermal load on ribs</i>
ItRibSelfWeight = 0x18,	<i>self weight load on ribs</i>
ItRibConcentrated = 0x19,	<i>concentrated load on ribs</i>
ItRibDistributed = 0x1A,	<i>distributed load on ribs</i>
ItSupportDisplacement = 0x1B,	<i>support displacement</i>
ItDomainConcentrated = 0x1C,	<i>concentrated load on domains</i>
ItSurfaceConcentrated = 0x1E,	<i>concentrated load on surface elements</i>
ItArea = 0x21,	<i>distributed area load</i>
ItDomainArea = 0x22,	<i>distributed area load on domains</i>
ItDomainFluid = 0x24,	<i>fluid load on domains</i>
ItSurfaceFluid = 0x25,	<i>fluid load on surface elements</i>
ItDomainPoly = 0x26,	<i>polyline load on domains</i>
ItSurfaceToBeam = 0x27,	<i>surface load distributed over beams</i>
ItDomainPolyAssoc = 0x28,	<i>associative edge load on domains</i>
ItSurfaceToBeamAssoc = 0x29,	<i>associative surface load distributed over beams</i>
ItNone = 0xFFFFFFFF }	<i>none of the above</i>

Load types.

```
enum ESurfaceDomainDistributionType = {
    sddtSurface = 0x00,          surface
    sddtProjected = 0x01 }      projected
```

Surface/domain load distribution type.

```
enum ESystem = {
    sysGlobal = 0x00,           global system
    sysLocal = 0x01,           local system
    sysReference = 0x02 }      by reference
```

Coordinate system of load components.

Error codes

```
enum ELoadsError = {
    leInvalidLineType = -100001    load type is not compatible with the line type
    leErrorAddingLoad = -100002    error when adding load
    leErrorSettingLoad = -100003   error when setting load
    leInvalidLoadType = -100004    invalid operation for this load type
    leNotValidLineTypeForThisLoad = -100005  invalid line type when distributing surface load on lines
    leErrorSettingLines = -100006   cannot set line list when distributing surface load on lines
    leErrorSettingPoly = -100007   cannot set the load polygon
```

Records / structures

Nodal loads

```
RLoadNodalForce = (
    double Fx, Fy, Fz      X, Y, Z force components [kN]
    double Mx, My, Mz      moment components about the X, Y, Z axis [kNm]
```

long **Referenceld** *If set to 0, load components are in global directions.
If 0 < Referenceld ≤ [AxisVMReferences.Count](#), the load direction is set by the reference. Referenceld is the index of reference according to [AxisVMReferences](#).*

Truss loads

)

RLoadTrussFault = (
double **DL** *fault in length*
)

RLoadTrussStress = (
double **Force** *tension/compression
if Force > 0, there is a tension at the endpoints
if Force < 0, there is a compression at the endpoints*
)

RLoadTrussThermal = (
double **Tref** *reference temperature*
double **T0** *actual truss temperature*
)

Beam loads

RLoadBeamConcentrated = (
double **Fgx, Fgy, Fgz** *x, y, z force components [kN]*
double **Mgx, Mgy, Mgz** *moment components about the x, y, z axis [kNm]*
double **Position** *if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length*
[ESystem](#) **SystemGLR** *coordinate system of load components
if SystemGLR = sysReference, only Fgx and Mgx are valid*
RLoadBeamDistributed = (
double **qx1, qy1, qz1** *x, y, z force components [kN/m] at the 1st point*
double **mx1, my1, mz1** *moment components about the x, y, z axis [kNm/m] at the 1st point*
double **qx2, qy2, qz2** *x, y, z force components [kN/m] at the 2nd point*
double **mx2, my2, mz2** *moment components about the x, y, z axis [kNm/m] at the 2nd point*
[ESystem](#) **SystemGLR** *coordinate system of load components*
double **Position1** *position of the 1st point
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length*
double **Position2** *position of the 2nd point with the same sign convention*
[EBeamRibDistributionType](#) **DistributionType** *distributed by length or projected*
BOOL **Trapezoid** *trapezoid load*
)

RLoadBeamFault = (
double **DL** *fault in length*
)

RLoadBeamInfluence = (
double **Dx, Dy, Dz** *relative displacements of the influence line load (-1, 0, 1)*
double **Rxx, Ryy, Rzz** *relative rotations of the influence line load (-1, 0, 1)*
double **Position** *if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length*
RLoadBeamStress = (
double **Force** *tension/compression
if Force > 0, tension is applied
if Force < 0, compression is applied*
)

RLoadBeamThermal = (
double **Tref** *reference temperature*
double **Tsup** *top cord temperature (in the Axis direction)*
double **Tinf** *bottom cord temperature (in the Axis direction)*
EAxis **Axis** *direction of temperature variation*
)

[EDistributionType](#) **RLoadSurfaceToBeam** = (
[EDistributionType](#) **DistributionType** *type of the distributed load (global / local / projected)*
double **Px, Py, Pz** *surface load intensity [kN/m²]*
)

[EDistributionType](#) **RLoadSurfaceToBeamAssoc** = (
[EDistributionType](#) **DistributionType** *a distributed load tipusa (global / local / projected)*
double **Px, Py, Pz** *surface load intensity [kN/m²]*
)

Rib loads

RLoadRibConcentrated = (
double **Fgx, Fgy, Fgz** *x, y, z force components [kN]*

double **Mgx, Mgy, Mgz** moment components about the x, y, z axis [kNm]
double **Position** if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length
[ESystem](#) **SystemGLR** coordinate system of load components
if SystemGLR = sysReference, only Fgx and Mgx are valid

RLoadRibDistributed = (

double **qx1, qy1, qz1** x, y, z force components [kN/m] at the 1st point
double **mx1, my1, mz1** moment components about the x, y, z axis [kNm/m] at the 1st point
double **qx2, qy2, qz2** x, y, z force components [kN/m] at the 2nd point
double **mx2, my2, mz2** moment components about the x, y, z axis [kNm/m] at the 2nd point
[ESystem](#) **SystemGLR** coordinate system of load components
double **Position1** position of the 1st point
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length
double **Position2** position of the 2nd point with the same sign convention
[EBeamRibDistributionType](#) **DistributionType** distributed by length or projected
BOOL Trapezoid trapezoid load
)

RLoadRibThermal = (

double **Tref** reference temperature
double **Tsup** top cord temperature (in the Axis direction)
double **Tinf** bottom cord temperature (in the Axis direction)
EAxis Axis direction of temperature variation
)

Surface loads

RLoadSurfaceDistributed = (

double **qx, qy, qz** x, y, z force components [kN/m²]
[ESystem](#) **SystemGLR** coordinate system of load components
[ESurfaceDomainDistributionType](#) **DistributionType** type of the distributed load
)

RLoadSurfaceEdge = (

double **qx1, qx2, qx3, qx4** x force components [kN/m] at the edges
double **qy1, qy2, qy3, qy4** y force components [kN/m] at the edges
double **qz1, qz2, qz3, qz4** z force components [kN/m] at the edges
[ESystem](#) **Sys1, Sys2, Sys3, Sys4** load coordinate systems on edges
BOOL EL1, EL2, EL3, EL4 True, if a load is applied on the edge connecting corner nodes
(For quadrilateral elements: 1 → 2, 2 → 3, 3 → 4, 4 → 1)
(For triangular elements: 1 → 2, 2 → 3, 3 → 1)
For triangular elements the fourth component is ignored.
[ESurfaceDomainDistributionType](#) **DT1, DT2, DT3, DT4** type of the distributed load
)

RLoadSurfaceFluid = (

[EAxis](#) **Direction** direction of fluid load variation
double **Coord1, Coord2** coordinate of the 1st and 2nd point
double **P1, P2** fluid load intensity at the 1st and 2nd point [kN/m²]
long **SurfaceId** surface element index
)

RLoadSurfaceThermal = (

double **Tref** reference temperature
double **Tsup** top temperature according to the local z direction
double **Tinf** bottom temperature according to the local z direction
)

Domain loads

RLoadDomainArea = (

[EDistributionType](#) **DistributionType** type of the distributed load (global / local / projected)
[ELoadDistributionType](#) **LoadDistributionType** type of the distributed load (constant / linear intensity)
[EAxis](#) **Component** load component
double **P1, P2, P3** load intensity at the load reference points [kN/m²]
double **x1, y1, z1** global coordinates of the 1st load reference point
double **x2, y2, z2** global coordinates of the 2nd load reference point
double **x3, y3, z3** global coordinates of the 3rd load reference point
double **x, y, z** load position
BOOL WindowLoad window load (load falling on an opening is distributed along the edges)
)

RLoadDomainConcentrated = (

double **Fx, Fy, Fz** x, y, z force components [kN]
double **Mx, My, Mz** moment components about the x, y, z axis [kNm]
double **x, y, z** global load position
[ESystem](#) **SystemGLR** coordinate system of load components

```

)
RLoadDomainDistributed = (
double qx, qy, qz           x, y, z force components [kN/m2]
ESystem SystemGLR       coordinate system of load components
ESurfaceDomainDistributionType DistributionType type of the distributed load
)
RLoadDomainFluid = (
EAxis Direction         direction of fluid load variation
double Coord1, Coord2    coordinate of the 1st and 2nd point
double P1, P2            fluid load intensity at the 1st and 2nd point [kN/m2]
long DomainId           domain index
)
RLoadDomainPoly = (
double px1, px2          start and end value of load intensity in x direction [kN/m]
double py1, py2          start and end value of load intensity in y direction [kN/m]
double pz1, pz2          start and end value of load intensity in z direction [kN/m]
double pm1, pm2          start and end value of the moment about the x axis [kNm/m]
EDistributionType DistributionType type of distributed load (global / local / projected)
double Nx, Ny, Nz        domain plane normal
)
RLoadDomainPolyAssoc = (
double px1, px2          start and end value of load intensity in x direction [kN/m]
double py1, py2          start and end value of load intensity in y direction [kN/m]
double pz1, pz2          start and end value of load intensity in z direction [kN/m]
double pm1, pm2          start and end value of the moment about the x axis [kNm/m]
EDistributionType DistributionType type of distributed load (global / local / projected)
long ElementId          index of the domain edge in AxisVMLines
)
RLoadDomainThermal = (
double Tref              reference temperature
double Tsup             top temperature according to the local z direction
double Tinf             bottom temperature according to the local z direction
)

```

Support loads

```

RLoadSupportDisplacement = (
double ex, ey, ez        support displacement in x, y, z directions
double fx, fy, fz        support rotation about the x, y, z axis
)

```

Functions

Call [GetLoad](#) to read load data. Call [SetLoad](#) to write load data. As the load record contains different fields for each load type it is necessary to use special COM functions to read or write load records.

How to read load data

1. Call the [GetLoad](#) function of the object.
2. Call the [SafeArrayAccessData](#) COM function to get the pointer to the array data.

```

HRESULT SafeArrayAccessData(
    SAFEARRAY* psa,                psa is LoadData obtained from GetLoad
    void** ppvData);              ppvData is a pointer to the array data

```

3. Typecast the pointer to the appropriate load record (struct) – e.g. [RLoadNodalForce](#) – and read the first (and only) element of the array.

4. After usage free the pointer by calling [SafeArrayUnaccessData](#).

```

HRESULT SafeArrayUnaccessData(
    SAFEARRAY* psa);              psa is LoadData obtained from GetLoad

```

5. Free the SAFEARRAY by calling [SafeArrayDestroy](#).

How to modify load data

1. Pick the appropriate GUID for the load type.

```

RID_RLoadNodalForce           : TGUID = '{F09D7969-81BD-4231-89DC-E7F70F94C4D1}';
RID_RLoadSupportDisplacement : TGUID = '{930336EE-1561-42A9-AE85-64F45914C1EF}';
RID_RLoadRibConcentrated     : TGUID = '{2984C228-E6E7-408B-A072-EB19B4079C71}';
RID_RLoadBeamConcentrated    : TGUID = '{DC5D9521-B0F7-481D-BE94-496D5351F919}';
RID_RLoadBeamInfluence       : TGUID = '{091F9059-8C14-4217-A4DD-1480CD8F2199}';
RID_RLoadRibDistributed      : TGUID = '{7E40BBEA-A491-4721-B47D-F59DE6F479FE}';
RID_RLoadBeamDistributed     : TGUID = '{B81A35ED-145E-4250-B918-60549118CD61}';
RID_RLoadRibThermal         : TGUID = '{2AC945B4-88E3-4320-9508-C28116271C95}';

```

```

RID_RLoadSurfaceThermal : TGUID = '{B3DB8C14-41D0-48F3-A368-6F64419AF32E}';
RID_RLoadDomainThermal : TGUID = '{C12C36A4-478E-46A8-B40B-8D273A1B2CC0}';
RID_RLoadBeamThermal : TGUID = '{DFA0B7F7-3438-46F3-8D06-C86B2C1673A8}';
RID_RLoadBeamStress : TGUID = '{5754C6B8-EAF3-4771-84E6-3096C1B1D5EE}';
RID_RLoadTrussStress : TGUID = '{E8D1B316-6C1E-43FD-ACB3-D9F53CAE3721}';
RID_RLoadBeamFault : TGUID = '{2770123F-A490-46CB-B6E9-7605C36E87B8}';
RID_RLoadTrussFault : TGUID = '{F3A7CD95-DDF9-4A83-8123-D68379F98F5B}';
RID_RLoadTrussThermal : TGUID = '{E06C0219-7880-48CC-A109-100FD0B52477}';
RID_RLoadBeamEnd : TGUID = '{A9143059-1D33-4783-88EC-26CB7EE8293F}';
RID_RLoadRibSelfweight : TGUID = '{DE14C046-B46B-4F58-BBCC-4405756D3D9B}';
RID_RLoadBeamSelfweight : TGUID = '{2CAA568D-5CFE-42F2-859E-EC50C6868B2A}';
RID_RLoadTrussSelfweight : TGUID = '{86BD6D68-7D38-433F-A874-AF39683778A9}';
RID_RLoadDomainSelfweight : TGUID = '{A79D2538-7DB9-4E50-B24A-F4E9E3AE39FD}';
RID_RLoadSurfaceSelfweight : TGUID = '{9E4B4580-9B5B-4805-9E08-1DDCE622C079}';
RID_RLoadSurfaceDistributed : TGUID = '{8AA571EE-1CC7-4EC7-A97D-35AA3E7B6A66}';
RID_RLoadDomainDistributed : TGUID = '{FCBC46D9-F4B5-46FB-9BD9-56A05A6823FE}';
RID_RLoadSurfaceEdge : TGUID = '{C544C757-74D8-4BC9-99A9-0FC0A0A249C2}';
RID_RLoadDomainEdge : TGUID = '{8F859468-19A2-4E24-9EAE-B765ABAB675E}';
RID_RLoadDomainConcentrated : TGUID = '{CCBF4716-42E4-4DC2-BB0E-C5F8713347AE}';
RID_RLoadSurfaceConcentrated : TGUID = '{39D35DBB-DE19-4E30-B8CC-9FAC5583C98E}';
RID_RLoadArea : TGUID = '{E78F3C96-BBC9-4A90-887C-7A1437AEE4B5}';
RID_RLoadDomainArea : TGUID = '{13A7B8C4-6CD8-4471-BF3D-160631DE070E}';
RID_RLoadDomainFluid : TGUID = '{23DD2E66-54A5-4794-95BD-933544445BEE}';
RID_RLoadSurfaceFluid : TGUID = '{FAA5355D-8015-48DD-AF55-741051930891}';
RID_RLoadSurfaceToBeam : TGUID = '{72640E58-9100-494C-94A7-7F302285C09F}';
RID_RLoadSurfaceToBeamAssoc : TGUID = '{EE2C3673-A9C5-4B71-AD05-3F84D24C9213}';
RID_RLoadDomainPoly : TGUID = '{864D5D92-F008-4CA3-8AEC-5933478F7D09}';
RID_RLoadDomainPolyAssoc : TGUID = '{4AE46345-A54E-4E20-8B8B-5F4C1708F4AB}';

```

2. Call **GetRecordInfoFromGUIDs** COM function to obtain the load record interface.

```

HRESULT GetRecordInfoFromGUIDs(
    REFGUID rGUIDTypeLib, AxisVM Type Library GUID
    ULONG uVerMajor, {2E9C7810-D11A-4C34-B104-752A29CD1505}
    ULONG uVerMinor, COM interface major version number = 2
    LCID lcid, COM interface minor version number = 0
    REFGUID rGUIDTypeInfo, = $409
    IRecordInfo **ppRecInfo); GUID of the load type read from the above table
    [out] pointer to the record info

```

3. Call **SafeArrayCreateEx** COM function to create a one dimensional array of VT_RECORDs with one element:

```

SAFEARRAY SafeArrayCreateEx(
    VARTYPE vt, vt = VT_RECORD (36)
    unsigned int cDims, cDims = 1
    SAFEARRAYBOUND* rgsabound, rgsabound
    unsigned long cElements = 1,
    long lLbound = 1,
    PVOID pvExtra ); ppRecInfo obtained from the previous function

```

4. Call **SafeArrayAccessData** COM function to get the pointer to the array data.

```

HRESULT SafeArrayAccessData(
    SAFEARRAY* psa, psa is the return value of SafeArrayCreateEx
    void HUGEP** ppvData); ppvData is a pointer to the array data

```

5. Typecast the pointer to the appropriate load record (struct) – e.g. RLoadNodalForce – and read the first (and only) element of the array. Modify field values.

6. After usage free the pointer by calling **SafeArrayUnaccessData**.

```

HRESULT SafeArrayUnaccessData(
    SAFEARRAY* psa); psa same as above

```

7. Call the **SetLoad** method of the object to write the modified data.

8. Free the SAFEARRAY by calling **SafeArrayDestroy**.

Nodal loads

```

long AddNodalForce ([in] long NodeId, [in] long CaseId,
[in] RLoadNodalForce Data)

```

NodeId node index
(0 < Index ≤ [AxisVMNodes.Count](#))

CaseId load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data load data

Creates a nodal load. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Truss loads

long **AddTrussFault** ([in] long **LineId**, [in] long **Caseld**,
[in] [RLoadTrussFault](#) **Data**)

LineId line index (truss)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data load data

Creates a "fault in length" on a truss. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddTrussSelfWeight** ([in] long **LineId**, [in] long **Caseld**)

LineId line index (truss)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Creates a self weight load on a truss. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddTrussStress** ([in] long **LineId**, [in] long **Caseld**,
[in] [RLoadTrussStress](#) **Data**)

LineId line index (truss)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data load data

Creates a tension/compression on a truss. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddTrussThermal** ([in] long **LineId**, [in] long **Caseld**,
[in] [RLoadTrussThermal](#) **Data**)

LineId line index (truss)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data load data

Creates a thermal load on a truss. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Beam loads

long **AddBeamConcentrated** ([in] long **LineId**, [in] long **Caseld**,
[in] [RLoadBeamConcentrated](#) **Data**)

LineId line index (beam)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data load data

Creates a concentrated load on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamDistributed** ([in] long **LineId**, [in] long **Caseld**,
[in] [RLoadBeamDistributed](#) **Data**)

LineId line index (beam)
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data *load data*

Creates a distributed load on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamFault** ([in] long **LineId**, [in] long **Caseld**, [in] [RLoadBeamFault](#) **Data**)

LineId *line index (beam)*
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld *load case index*
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data *load data*

Creates a "fault in length" on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamInfluence** ([in] long **LineId**, [in] long **Caseld**, [in] [RLoadBeamInfluence](#) **Data**)

LineId *line index (beam)*
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld *load case index*
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data *load data*

Creates an influence line load on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamSelfWeight** ([in] long **LineId**, [in] long **Caseld**)

LineId *line index (beam)*
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld *load case index*
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Creates a self weight load on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamStress** ([in] long **LineId**, [in] long **Caseld**, [in] [RLoadBeamStress](#) **Data**)

LineId *line index (beam)*
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld *load case index*
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data *load data*

Creates tension/compression on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddBeamThermal** ([in] long **LineId**, [in] long **Caseld**, [in] [RLoadBeamThermal](#) **Data**)

LineId *line index (beam)*
(0 < Index ≤ [AxisVMLines.Count](#))

Caseld *load case index*
(0 < Index ≤ [AxisVMLoadCases.Count](#))

Data *load data*

Creates a thermal load on a beam. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceToBeam** ([in] [AxisVMLines3d](#) **ContourPoly**, [in] **BOOL** **AutoFindLines**, [in] **SAFEARRAY(long)*** **Linelds**, [in] long **Caseld**, [in] [RLoadSurfaceToBeam](#) **Data**)

ContourPoly closed polygon of the surface load to distribute

AutoFindLines if set to True, AxisVM determines the line elements participating in the distribution process automatically. If set to False, the surface load is distributed over line elements specified in the Linelds array.

Linelds line indices of those lines, beams, ribs, trusses or rigid bodies which participate in the distribution process. If AutoFindLines is True, this parameter is ignored. See [IAxisVMLines](#)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Distributes a homogenous surface load defined by a closed polygon over line elements. If successful, returns load index, otherwise returns the error code ([leNotValidLineTypeForThisLoad](#), [leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceToBeamAssoc** ([in] **SAFEARRAY(long)*** **ContourLinelds**, [in] **BOOL** **AutoFindLines**, [in] **SAFEARRAY(long)*** **Linelds**, [in] long **Caseld**, [in] [RLoadSurfaceToBeamAssoc](#) **Data**)

ContourLinelds line indices of contour lines of the closed polygon

AutoFindLines if set to True, AxisVM determines the line elements participating in the distribution process automatically. If set to False, the surface load is distributed over line elements specified in the Linelds array.

Linelds line indices of those lines, beams, ribs, trusses or rigid bodies which participate in the distribution process. If AutoFindLines is True, this parameter is ignored. See [IAxisVMLines](#)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Distributes an associative surface load defined by a closed polygon made of existing lines over line elements. This surface load shape follows changes in model geometry. If successful, returns load index, otherwise returns the error code ([leNotValidLineTypeForThisLoad](#), [leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Rib loads

long **AddRibConcentrated** ([in] long **Lineld**, [in] long **Caseld**, [in] [RLoadRibConcentrated](#) **Data**)

Lineld line index (rib)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a concentrated load on a rib. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddRibDistributed** ([in] long **Lineld**, [in] long **Caseld**, [in] [RLoadRibDistributed](#) **Data**)

Lineld line index (rib)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a distributed load on a rib. If successful, returns load index, otherwise returns

the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddRibSelfWeight** ([in] long **Lineld**, [in] long **Caseld**)

Lineld line index (rib)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Creates a self weight load for a rib. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddRibThermal** ([in] long **Lineld**, [in] long **Caseld**,
[in] [RLoadRibThermal](#) Data)

Lineld line index (rib)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a thermal load on a rib. If successful, returns load index, otherwise returns the error code ([leInvalidLineType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Surface loads

long **AddSurfaceDistributed** ([in] long **Surfaceld**, [in] long **Caseld**,
[in] [RLoadSurfaceDistributed](#) Data)

Surfaceld surface element index
($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a distributed load on a surface element. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceEdge** ([in] long **Surfaceld**, [in] long **Caseld**,
[in] [RLoadSurfaceEdge](#) Data)

Surfaceld surface element index
($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates an edge load on a surface element. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceFluid** ([in] long **Caseld**, [in] [RLoadSurfaceFluid](#) Data)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a fluid load on a surface element. If successful, returns load index, otherwise returns the error code ([leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceSelfWeight** ([in] long **Surfaceld**, [in] long **Caseld**)

Surfaceld surface element index
($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Creates a self weight load on a surface element. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddSurfaceThermal** ([in] long **SurfaceId**, [in] long **Caseld**, [in] [RLoadSurfaceThermal](#) **Data**)

SurfaceId *surface element index*
($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)
Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a thermal load on a surface element. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Domain loads

long **AddArea** ([in] [AxisVMLines3d](#) **ContourPoly**, [in] long **Caseld**, [in] [RLoadDomainArea](#) **Data**)

ContourPoly *a closed polygon*
Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a mesh-independent distributed load on a domain. If successful, returns load index, otherwise returns the error code ([leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainConcentrated** ([in] long **DomainId**, [in] long **Caseld**, [in] [RLoadDomainConcentrated](#) **Data**)

DomainId *domain index*
($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)
Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a mesh-independent concentrated load on a domain. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainDistributed** ([in] long **DomainId**, [in] long **Caseld**, [in] [RLoadDomainDistributed](#) **Data**)

DomainId *domain index*
($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)
Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a distributed load on a domain. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainFluid** ([in] long **Caseld**, [in] [RLoadDomainFluid](#) **Data**)

Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a fluid load on a domain. If successful, returns load index, otherwise returns the error code ([leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainPoly** ([in] [AxisVMLines3d](#) **LoadPoly**, [in] long **Caseld**, [in] [RLoadDomainPoly](#) **Data**)

LoadPoly *the load polygon*
Caseld *load case index*
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
Data *load data*

Creates a mesh-independent polyline load on a domain. If successful, returns load index, otherwise returns the error code ([leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainPolyAssoc** ([in] long **Caseld**,
[in] [RLoadDomainPolyAssoc](#) **Data**)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates an associative line load on a domain edge. If successful, returns load index, otherwise returns the error code ([leErrorAddingLoad](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainSelfWeight** ([in] long **DomainId**, [in] long **Caseld**)

DomainId domain index
($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Creates a self weight on a domain. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **AddDomainThermal** ([in] long **DomainId**, [in] long **Caseld**,
[in] [RLoadDomainThermal](#) **Data**)

DomainId domain index
($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a thermal load on a domain. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Support loads

long **AddSupportDisplacement** ([in] long **SupportId**, [in] long **Caseld**,
[in] [RLoadSupportDisplacement](#) **Data**)

SupportId nodal support index
($0 < \text{Index} \leq \text{AxisVMNodalSupports.Count}$)

Caseld load case index
($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)

Data load data

Creates a support displacement. If successful, returns load index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Other functions

long **Delete** ([in] long **Index**)

Index index of the load to delete ($0 < \text{Index} \leq \text{Count}$)

Deletes a load from the model.
If successful, returns *Index*, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **GetLines** ([in] long **Index**, [out] SAFEARRAY(long)* **Linelds**)

Index load index ($0 < \text{Index} \leq \text{Count}$)

Linelds lines participating in the distribution process according to [AxisVMLines](#)

(only for *ItSurfaceToBeam* or *ItSurfaceToBeamAssoc*) Obtains the line indices participating in the distribution process. If successful, returns *Index*, otherwise returns the error code ([leInvalidLoadType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **GetLoad** ([in] long **Index**, [out] SAFEARRAY* **LoadData**)

Index load index ($0 < \text{Index} \leq \text{Count}$)

LoadData a pointer to the load record in SAFEARRAY format

Reads a load record. If successful, returns Index, otherwise returns the error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#), or positive values).

long **GetPoly** ([in] long Index, [out] [AxisVMLines3d](#)* Poly)
Index load index ($0 < \text{Index} \leq \text{Count}$)
Poly the load polygon. For *ItArea* and *ItSurfaceToBeam* types it is a closed polygon, for *ItDomainPoly* it is a polyline.
(only for *ItArea*, *ItDomainPoly*, *ItSurfaceToBeam* types) Reads the load polygon. If successful, returns Index, otherwise returns the error code ([leInvalidLoadType](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#))

long **SetLines** ([in] long Index, [in] SAFEARRAY(long)* LineIds)
Index load index ($0 < \text{Index} \leq \text{Count}$)
LineIds lines participating in the distribution process according to [AxisVMLines](#)
(only for *ItSurfaceToBeam* or *ItSurfaceToBeamAssoc*) Sets the line indices participating in the distribution process. If successful, returns Index, otherwise returns the error code ([leInvalidLoadType](#), [leErrorSettingLines](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#))

long **SetLoad** ([in] long Index, [in] SAFEARRAY* LoadData)
Index load index ($0 < \text{Index} \leq \text{Count}$)
LoadData a pointer to the load record in SAFEARRAY format
Reads a load record. If successful, returns Index, otherwise returns the error code (can be positive).

long **SetPoly** ([in] long Index, [in] [AxisVMLines3d](#)* Poly)
Index load index ($0 < \text{Index} \leq \text{Count}$)
Poly the load polygon. For *ItArea* and *ItSurfaceToBeam* types it is a closed polygon, for *ItDomainPoly* it is a polyline.
(only for *ItArea*, *ItDomainPoly*, *ItSurfaceToBeam* types) Sets the load polygon. If successful, returns Index, otherwise returns the error code ([leInvalidLoadType](#), [leErrorSettingPoly](#), [errIndexOutOfBounds](#), [errDatabaseNotReady](#))

Properties

long **Caseld** [long Index] •
load case index of a load

long **Count**
Number of loads in the model

long **ElementId** [long Index]
Index of the element associated to a load (node, line, surface, domain, etc.).

[ELoadType](#) **LoadType** [long Index]
Type of a load.

IAxisVMMaterials

Materials of the model.

Error codes

```
enum EMaterialError = {  
    meEmpty_Name = -100001           material does not have a name  
    meNegative_Nux = -100002          $v_x < 0$   
    meNegative_Nuy = -100003          $v_y < 0$   
    meNegative_Nuz = -100004          $v_z < 0$   
    meGreaterThan05_Nux = -100005     $v_x > 0,5$   
    meGreaterThan05_Nuy = -100006     $v_y > 0,5$   
    meGreaterThan05_Nuz = -100007     $v_z > 0,5$ 
```

meNegative_Alphax = -100008	$\alpha_x < 0$	
meNegative_Alphay = -100009	$\alpha_y < 0$	
meNegative_Alphaz = -100010	$\alpha_z < 0$	
meNonPositive_Ex = -100011	$E_x \leq 0$	
meNonPositive_Ey = -100012	$E_y \leq 0$	
meNonPositive_Ez = -100013	$E_z \leq 0$	
meNonPositive_Rho = -100014	$\rho \leq 0$	
meNonPositive_SigmaH = -100015	$\sigma_H \leq 0$	MSz steel
meNonPositive_SigmaPH = -100016	$\sigma_{PH} \leq 0$	MSz steel
meNonPositive_Ry = -100017	$R_y \leq 0$	MSz steel
meNonPositive_Fy = -100018	$f_y \leq 0$	EC, I, DIN, SIA steel
	$f_{yd} \leq 0$	NEN steel
meNonPositive_Fu = -100019	$f_u \leq 0$	EC, I, DIN, SIA steel
	$f_{yt} \leq 0$	NEN steel
meNonPositive_Fy40 = -100020	$f_y^* \leq 0$	EC, I, DIN, SIA steel
	$f_{yd}^* \leq 0$	NEN steel
meNonPositive_Fu40 = -100021	$f_u^* \leq 0$	EC, I, DIN, SIA steel
	$f_{yt}^* \leq 0$	NEN steel
meNonPositive_R = -100022	$R \leq 0$	STAS steel
meNonPositive_Rc = -100023	$R_c \leq 0$	STAS steel
meNonPositive_SigmaBH = -100024	$\sigma_{BH} \leq 0$	MSz concrete
	$R_{bc} \leq 0$	STAS concrete
meNonPositive_SigmaBH = -100025	$\sigma_{BH} \leq 0$	MSz concrete
	$R_{bt} \leq 0$	STAS concrete
meNonPositive_Fit = -100026	$\Phi \leq 0$	STAS, NEN concrete
	$\Phi_k \leq 0$	EC, I, DIN, SIA concrete
meNonPositive_Fck = -100027	$f_{ck} \leq 0$	EC, I, DIN, SIA concrete
	$f_{ck}^t \leq 0$	NEN concrete
meNonPositive_GammaC = -100028	$\gamma_c \leq 0$	EC, I, DIN, SIA concrete
meNonPositive_Alphacc = -100029;	$\alpha_{cc} \leq 0$	EC, I concrete
	$\alpha \leq 0$	DIN concrete
meNonPositive_Fck_cube = -100030	$f_{ck, cube} \leq 0$	DIN concrete
meInvalid_MaterialType = -100031		material type is invalid
meInvalid_NationalDesignCode = -100032		national design code is invalid
meNameAlreadyExists = -100033		material name already exists
}		

Functions

New materials can be added to the model calling [AddDialog](#) (displays an AxisVM dialog to get material parameters) or [AddFromCatalog](#) (reads a material from the catalog). New materials can be created by calling any of the [Add...](#) functions.

FillColor and **ContourColor** must be specified as a 4 byte unsigned long values. The first byte must be 0, the other three bytes represent blue, green and red values. So 0x00000000 is black, 0x00FF0000 is blue, 0x0000FF00 is green, 0x000000FF is red, 0x00FFFFFF is white.

The first 15 parameters of all **Add...** functions are common regardless the type or national design code of the material:

```
long Add... ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColor,
[in] unsigned long ContourColor, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alphax, [in] double
Alphay, [in] double Alphaz, [in] double Rho, ...)
```

NationalDesignName *name of the national design code*

MaterialDesignName *name of the material code*

Name *name of the material*

FillColor *fill color used in rendered view*

ContourColor *outline color used in rendered view*

Ex, Ey, Ez *Young's modulus of elasticity in local directions [kN/m²]*

Nux, Nuy, Nuz *ν Poisson's ratio in local directions $0 \leq \nu \leq 0,5$*

Alphax, Alphay, Alphaz *α thermal expansion coefficient in local directions [1/°C]*

Rho

ρ density [kg/m³]

Adds a new model to the material. If successful, returns the material index, otherwise returns the error code ([errDatabaseNotReady](#) or [EMaterialError](#) codes).

Other parameters may change according to the material type and the design code. Extra parameters are listed in the function description.

long **AddAluminium** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**)

For parameters see [Add...](#)

Adds an aluminium material to the model.

long **AddBrick** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**)

For parameters see [Add...](#)

Adds a brick material to the model.

long **AddConcrete_Dutch_NEN** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fck**,
[in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

Fck f'_{ck} characteristic compressive cylinder strength at 28 days
[kN/m²]

Fit Φ creeping factor

Adds a concrete according to the Dutch code to the model.

long **AddConcrete_Eurocode** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fck**,
[in] double **GammaC**, [in] double **Alphacc**, [in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

Fck f_{ck} characteristic compressive cylinder strength at 28 days
[kN/m²]

GammaC γ_c safety factor of the concrete

Alphacc α_{cc} concrete strength-reduction factor for sustained loading

Fit Φ_t creeping factor

Adds a concrete according to Eurocode to the model.

long **AddConcrete_German_DIN1045_1** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fck**,
[in] double **Fck_cube**, [in] double **GammaC**, [in] double **Alphacc**, [in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

Fck f_{ck} characteristic compressive cylinder strength at 28 days
[kN/m²]

Fck_cube $f_{ck, cube}$ characteristic compressive cylinder strength of cube
[kN/m²]

GammaC γ_c safety factor of the concrete
Alphacc α concrete strength-reduction factor for sustained loading
Fit Φ_t creeping factor

Adds a concrete according to DIN 1045-1 to the model.

long **AddConcrete_Hungarian_MSz** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **SigmabH**, [in] double **SigmahH**, [in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

SigmabH σ_{bh} resistance for compression [kN/m²]
SigmahH σ_{hh} resistance for tension [kN/m²]
Fit Φ creeping factor

Adds a concrete according to the Hungarian code to the model.

long **AddConcrete_Italian** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fck**, [in] double **GammaC**, [in] double **Alphacc**, [in] double **Fit**)

See [AddConcrete_EuroCode](#)

Adds a concrete according to the Italian code to the model.

long **AddConcrete_Romanian_STAS** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **SigmabH**, [in] double **SigmahH**, [in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

SigmabH R_{bc} resistance for compression [kN/m²]
SigmahH R_{bt} resistance for tension [kN/m²]
Fit Φ creeping factor

Adds a concrete according to the Romanian code to the model.

long **AddConcrete_Swiss_SIA26x** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fck**, [in] double **GammaC**, [in] double **Fit**)

For the beginning of the parameter list see [Add...](#)

Fck f_{ck} characteristic compressive cylinder strength at 28 days [kN/m²]
GammaC γ_c safety factor of the concrete
Fit Φ_t creeping factor

Adds a concrete according to the Swiss code to the model.

long **AddDialog** ([in] ENationalDesignCode **NationalDesignCode**)

NationalDesignCode national design code of the material

Adds a material of the given national design code to the model by displaying an AxisVM dialog to enter parameters.

If successful, returns the material index. If the Cancel button was clicked returns 0

otherwise returns an error code.

long **AddFromCatalog** ([in] ENationalDesignCode **NationalDesignCode**,
[in] BSTR **MaterialName**)

NationalDesignCode national design code of the material

MaterialName name of the material

Adds a material from the catalog to the model.

If succesful, returns the material index, otherwise returns an error code.

long **AddSteel_Dutch_NEN** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fy**, [in] double **Fu**,
[in] double **Fy40**, [in] double **Fu40**)

Fy f_{yd} yield stress [kN/m²]

Fu f_{yt} ultimate stress [kN/m²]

Fy40 f_{yd}^* yield stress
if thickness is between 40 and 100 mm [kN/m²]

Fu40 f_{yt}^* ultimate stress
if thickness is between 40 and 100 mm [kN/m²]

Adds a steel according to the Dutch code to the model.

long **AddSteel_EuroCode** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fy**, [in] double **Fu**,
[in] double **Fy40**, [in] double **Fu40**)

For the beginning of the parameter list see [Add...](#)

Fy f_y yield stress [kN/m²]

Fu f_u ultimate stress [kN/m²]

Fy40 f_y^* yield stress
if thickness is between 40 and 100 mm [kN/m²]

Fu40 f_u^* ultimate stress
if thickness is between 40 and 100 mm [kN/m²]

Adds a steel according to Eurocode to the model.

long **AddSteel_German_DIN1045_1** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fy**, [in] double **Fu**,
[in] double **Fy40**, [in] double **Fu40**)

See [AddSteel_EuroCode](#)

Adds a steel according to DIN 1045-1 to the model.

long **AddSteel_Hungarian_MSz** ([in] BSTR **NationalDesignName**,
[in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**,
[in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**,
[in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**,
[in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **SigmaH**,
[in] double **SigmapH**, [in] double **Ry**)

For the beginning of the parameter list see [Add...](#)

SigmaH σ_H resistance [kN/m²]

SigmapH σ_{pH} cylinder-jacket resistance [kN/m²]

Ry *R_y ultimate stress [kN/m²]*

Adds a steel according to the Hungarian code to the model.

long **AddSteel_Italian** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fy**, [in] double **Fu**, [in] double **Fy40**, [in] double **Fu40**)

See [AddSteel_EuroCode](#)

Adds a steel according to the Italian code to the model.

long **AddSteel_Romanian_STAS** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **R**, [in] double **Rc**)

For the beginning of the parameter list see [Add...](#)

R *R ultimate stress [kN/m²]*

Rc *Rc resistance [kN/m²]*

Adds a steel according to the Romanian code to the model.

long **AddSteel_Swiss_SIA26x** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**, [in] double **Fy**, [in] double **Fu**, [in] double **Fy40**, [in] double **Fu40**)

See [AddSteel_EuroCode](#)

Adds a steel according to the Swiss code to the model.

long **AddTimber** ([in] BSTR **NationalDesignName**, [in] BSTR **MaterialDesignName**, [in] BSTR **Name**, [in] unsigned long **FillColor**, [in] unsigned long **ContourColor**, [in] double **Ex**, [in] double **Ey**, [in] double **Ez**, [in] double **Nux**, [in] double **Nuy**, [in] double **Nuz**, [in] double **Alphax**, [in] double **Alphay**, [in] double **Alphaz**, [in] double **Rho**)

For parameters see [Add...](#)

Adds a timber material to the model.

long **Clear**
Deletes all materials of the model. If successful, returns the number of deleted materials, otherwise an error code.

long **Delete** ([in] long **Index**)
Index *index of the material to delete (0 < Index ≤ Count)*

*Deletes a material from the model
If successful, returns Index, otherwise an error code.*

long **IndexOf** ([in] BSTR **Name**)
Name *name of the material*
Finds a material by its name. If successful, returns the index of the material, otherwise an error code ([errDatabaseNotReady](#) or [errNotFound](#)).

Properties

long **Count**

Number of materials in the model.

[AxisVMMaterial*](#)

Item [long Index]

Material object by index. $1 \leq \text{Index} \leq \text{Count}$.

IAxisVMMaterial

Material object in the model or catalog.

Enumerated types

```
enum EMaterialType = {  
    mtOther = 0x0,          other  
    mtSteel = 0x1,         steel  
    mtConcrete = 0x2,      concrete  
    mtTimber = 0x3,        timber  
    mtAluminium = 0x4,     aluminium  
    mtBrick = 0x5 }  
Material type.
```

Properties

double **Alphax** • α thermal expansion coefficient in local x direction [1/°C]
double **Alphay** • α thermal expansion coefficient in local y direction [1/°C]
double **Alphaz** • α thermal expansion coefficient in local z direction [1/°C]
unsigned long **ContourColor** • outline color used in rendered view (see [IAxisVMMaterials](#))
double **Ex** • Young's modulus of elasticity in local x direction [kN/m²]
double **Ey** • Young's modulus of elasticity in local y direction [kN/m²]
double **Ez** • Young's modulus of elasticity in local z direction [kN/m²]
unsigned long **FillColor** • fill color used in rendered view (see [IAxisVMMaterials](#))
BSTR **MaterialDesignName** • name of the material code
[EMaterialType](#) **MaterialType** • type of the material
BSTR **Name** • material name
[ENationalDesignCode](#) **NationalDesignCode** • national design code
BSTR **NationalDesignName** • name of the national design code
double **Nux** • ν Poisson's ratio in local x direction $0 \leq \nu \leq 0,5$
double **Nuy** • ν Poisson's ratio in local y direction $0 \leq \nu \leq 0,5$
double **Nuz** • ν Poisson's ratio in local z direction $0 \leq \nu \leq 0,5$
double **Rho** • ρ density [kg/m³]

Design parameters of steel materials

EC, I, DIN, SIA, NEN

double **Fu** • ultimate stress
(EC, DIN, I, SIA: f_u) (NEN: f_{yt}) [kN/m²]
double **Fu40** • ultimate stress if thickness is between 40 and 100 mm
(EC, DIN, I, SIA: f_u^*) (NEN: f_{yt}^*) [kN/m²]
double **Fy** • yield stress
(EC, DIN, I, SIA: f_y) (NEN: f_{yd}) [kN/m²]
double **Fy40** • yield stress if thickness is between 40 and 100 mm
(EC, DIN, I, SIA: f_y^*) (NEN: f_{yd}^*) [kN/m²]

MSz

double **Ry** • ultimate stress (R_y) [kN/m²]
double **SigmaH** • resistance (σ_H) [kN/m²]

double **SigmapH** • cylinder-jacket resistance (σ_{pH}) [kN/m^2]

STAS

double **R** • ultimate stress (R) [kN/m^2]

double **Rc** • yield stress (R_c) [kN/m^2]

Design parameters of concrete materials

double **Alphacc** • concrete strength-reduction factor for sustained loading (EC, I: α_{cc}) (DIN: α)

double **Fck** • characteristic compressive cylinder strength at 28 days (EC, I, DIN, SIA: f_{ck}) (NEN: f_{ck}') [kN/m^2]

double **Fck_cube** • characteristic compressive cylinder strength of cube (DIN: $f_{ck,cube}$) [kN/m^2]

double **Fit** • creeping factor (EC, I, DIN, SIA: Φ_i) (NEN, STAS: Φ)

double **Gammac** • safety factor of the concrete (EC, I, DIN, SIA: γ_c)

double **SigmabH** • resistance for compression (MSz: σ_{bH} , STAS: R_{bc}) [kN/m^2]

double **SigmahH** • resistance for tension (MSz: σ_{tH} , STAS: R_{bt}) [kN/m^2]

IAxisVMNodalSupports

Nodal supports in the model.

Error codes

```
enum ESupportError = {  
    seNodeIndexOutOfBounds = -100001    node index  $\leq 0$  or  $\geq$  AxisVMNodes.Count  
    seLineIndexOutOfBounds = -100002    line index  $\leq 0$  or  $\geq$  AxisVMLines.Count  
    seReferenceIndexOutOfBounds = -100003 } reference index  $\leq 0$  or  $\geq$  AxisVMReferences.Count
```

Records / structures

```
RNonlinearity = (  
    ELineNonlinearity x, y, z,    nonlinear behaviour in x, y, z directions  
                    xx, yy, zz    and for rotation about the x, y, z axis  
                    )  
  
RResistances = (  
    double x, y, z,    resistance in x, y, z directions [kN]  
                    xx, yy, zz    moment resistances about the x, y, z axis [kNm]  
                    )  
  
RStiffnesses = (  
    double x, y, z    stiffness in x, y, z direction [kN/m]  
    double xx, yy, zz    rotational stiffness around x, y, z axes [kNm/rad]  
                    )
```

Functions

```
long AddNodalBeamRelative ([in] RStiffnesses Stiffnesses,  
    [in] RNonLinearity NonLinearity, [in] RResistances Resistances, [in] long NodeId,  
    [in] long BeamId)
```

Stiffnesses nodal support stiffnesses
NonLinearity nonlinear behaviour of support components
Resistances resistance of support components
NodeId node index
($0 < \text{Index} \leq \text{AxisVMNodes.Count}$)
BeamId line index (beam or rib)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)

Adds a nodal support to the model. Components are defined in the local system of the beam or rib. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seLineIndexOutOfBounds](#)).

long **AddNodalEdgeRelative** ([in] [RStiffnesses](#) Stiffnesses, [in] [RNonLinearity](#) NonLinearity, [in] [RResistances](#) Resistances, [in] long Nodeld, [in] long Lineld, [in] long Surfaced1, [in] long Surfaced2, [in] long DomainId1, [in] long DomainId2,)

Stiffnesses nodal support stiffnesses
NonLinearity nonlinear behaviour of support components
Resistances resistance of support components
Nodeld node index
($0 < \text{Index} \leq \text{AxisVMNodes.Count}$)
Lineld line index (edge)
($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
Surfaced1 first connecting surface
($0 < \text{Surfaced1} \leq \text{AxisVMSurfaces.Count}$)
Surfaced2 second connecting surface
($0 < \text{Surfaced2} \leq \text{AxisVMSurfaces.Count}$)
DomainId1 first connecting domain
($0 < \text{DomainId1} \leq \text{AxisVMDomains.Count}$)
DomainId2 second connecting domain
($0 < \text{DomainId2} \leq \text{AxisVMDomains.Count}$)

Adds a nodal support to the model. Connecting surfaces and domains are stored so that if all of them is deleted the support is deleted as well. Components are defined in a local system determined by the edge and the surface local z direction. If only one connecting surface or domain is specified (other surface/domain indices are zero) the local x direction is along the edge, the local y direction is in the plane of the surface and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the surface. If two indices are nonzero the local x direction is along the edge, the local z direction is on the bisector of the local z directions of the two, the local y direction is perpendicular to both. If successful, returns the support index, otherwise returns the error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seLineIndexOutOfBounds](#)).

long **AddNodalGlobal** ([in] [RStiffnesses](#) Stiffnesses, [in] [RNonLinearity](#) NonLinearity, [in] [RResistances](#) Resistances, [in] long Nodeld)

Stiffnesses nodal support stiffnesses
NonLinearity nonlinear behaviour of support components
Resistances resistance of support components
Nodeld node index
($0 < \text{Index} \leq \text{AxisVMNodes.Count}$)

Adds a nodal support to the model. Components are defined in the global directions. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#)).

long **AddNodalReference** ([in] [RStiffnesses](#) Stiffnesses, [in] [RNonLinearity](#) NonLinearity, [in] [RResistances](#) Resistances, [in] long Nodeld, [in] long Referenceld)

Stiffnesses nodal support stiffnesses
NonLinearity nonlinear behaviour of support components
Resistances resistance of support components
Nodeld node index
($0 < \text{Index} \leq \text{AxisVMNodes.Count}$)
Referenceld a reference index
($0 < \text{Index} \leq \text{AxisVMReferences.Count}$)

Adds a nodal support to the model. Support direction is defined by the reference. Only x and xx components are taken into account. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seReferenceIndexOutOfBounds](#)).

long **Delete** ([in] long Index)

Deletes a nodal support. $1 \leq \text{Index} \leq \text{Count}$.
If successful, returns the number of deleted supports, otherwise returns an error code

[\(errDatabaseNotReady, errIndexOutOfBounds\)](#).

long **DeleteSelected**
Deletes the selected nodal supports. If successful, returns the number of deleted supports, otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

long **Count** *number of nodal supports in the model*
[AxisVMNodalSupport*](#) **Item** [long **Index**] *a nodal support object*
BOOL **Selected** [long **Index**] • *gets or sets the selection status of a nodal support*
long **SelCount** *number of selected nodal supports in the model*

IAxisVMNodalSupport

AxisVM nodal support object.

Enumerated types

enum **ENodalSupportType** = {
 nstNodalGlobal = 0x00, *nodal support in global directions*
 nstNodalBeamRelative = 0x01, *beam / rib relative nodal support*
 nstNodalEdgeRelative = 0x02, *edge relative nodal support*
 nstNodalReference = 0x03 } *nodal support in reference direction*
Nodal support types

Error codes

enum **ENodalSupportError** = {
 nsePropertyNotValidForThisType = -100001 } *property is not compatible with the support type*
enum **ESupportError** = {
 seNodeIndexOutOfBounds = -100001 *node index ≤ 0 or \geq [AxisVMNodes.Count](#)*
 seLineIndexOutOfBounds = -100002 *line index ≤ 0 or \geq [AxisVMLines.Count](#)*
 seReferenceIndexOutOfBounds = -100003 } *reference index ≤ 0 or \geq [AxisVMReferences.Count](#)*

Records / structures

RNonlinearity = (
 ELineNonlinearity **x, y, z,** *nonlinear behaviour in x, y, z directions*
 xx, yy, zz *and for rotation about the x, y, z axis*
)

RResistances = (
 double **x, y, z,** *resistance in x, y, z directions [kN]*
 xx, yy, zz *moment resistances about the x, y, z axis [kNm]*
)

RStiffnesses = (
 double **x, y, z** *stiffness in x, y, z direction [kN/m]*
 double **xx, yy, zz** *rotational stiffness around x, y, z axes [kNm/rad]*
)

Functions

long **DefineAsNodalBeamRelative** ([in] [RStiffnesses](#) **Stiffnesses**,
[in] [RNonLinearity](#) **NonLinearity**, [in] [RResistances](#) **Resistances**, [in] long **NodeId**,
[in] long **BeamId**)
Redefines a support. For parameters see [IAxisVMNodalSupports.AddNodalBeamRelative](#).

long **DefineAsNodalEdgeRelative** ([in] [RStiffnesses](#) **Stiffnesses**,
[in] [RNonLinearity](#) **NonLinearity**, [in] [RResistances](#) **Resistances**, [in] long **NodeId**,
[in] long **LineId**, [in] long **Surfaceld1**, [in] long **Surfaceld2**, [in] long **DomainId1**,
[in] long **DomainId2**,)
Redefines a support. For parameters see

[IAxisVMNodalSupports.AddNodalEdgeRelative.](#)

long **DefineAsNodalGlobal** ([in] [RStiffnesses](#) **Stiffnesses**, [in] [RNonLinearity](#) **NonLinearity**, [in] [RResistances](#) **Resistances**, [in] long **NodeId**)

Redefines a support. For parameters see [IAxisVMNodalSupports.AddNodalGlobal.](#)

long **DefineAsNodalReference** ([in] [RStiffnesses](#) **Stiffnesses**, [in] [RNonLinearity](#) **NonLinearity**, [in] [RResistances](#) **Resistances**, [in] long **NodeId**, [in] long **ReferenceId**)

Redefines a support. For parameters see [IAxisVMNodalSupports.AddNodalReference.](#)

Properties

If **BeamId**, **DomainId...**, **LineId**, **ReferenceId** or **SurfaceId...** property is not compatible with the support type, its value is *nsePropertyNotValidForThisType*.

long **BeamId** (if *SupportType* is *nstNodalBeamRelative*) beam / rib index

long **DomainId1** (if *SupportType* is *nstNodalEdgeRelative*) 1st connecting domain

long **DomainId2** (if *SupportType* is *nstNodalEdgeRelative*) 2nd connecting domain

long **LineId** (if *SupportType* is *nstNodalEdgeRelative*) edge index

long **NodeId** node index

[RNonLinearity](#) **NonLinearity** • nonlinear behaviour of support components

long **ReferenceId** (if *SupportType* is *nstNodalReference*) a reference index

[RResistances](#) **Resistances** • resistances of support components

[RStiffnesses](#) **Stiffnesses** • support stiffnesses

[ENodalSupportType](#) **SupportType** support type

long **SurfaceId1** (if *SupportType* is *nstNodalEdgeRelative*) 1st connecting surface

long **SurfaceId2** (if *SupportType* is *nstNodalEdgeRelative*) 2nd connecting surface

IAxisVMNodes

Nodes in the model.

Enumerated types

enum **EDegreeOfFreedom** = {
 dofXfix = 0x01, *no displacement in X direction*
 dofYfix = 0x02, *no displacement in Y direction*
 dofZfix = 0x04, *no displacement in Z direction*
 dofXXfix = 0x08, *no rotation in X direction*
 dofYYfix = 0x10, *no rotation in Y direction*
 dofZZfix = 0x20 } *no rotation in Z direction*
DOF components. Combine constraints by adding the constants.

Konstansok:

long **dofTrussAndMembraneXZ** = 0x3A
(= 0x20 + 0x10 + 0x08 + 0x02)
(= dofZZfix + dofYYfix + dofXXfix + dofYfix)
Degrees of freedom for a membrane model in X-Z plane

long **dofFrameXZ** = 0x2A
(= 0x20 + 0x08 + 0x02)
(= dofZZfix + dofXXfix + dofYfix)
Degrees of freedom for a frame model in X-Z plane

long **dofPlaneXY** = 0x23
(= 0x20 + 0x02 + 0x01)
(= dofZZfix + dofYfix + dofXfix)

Degrees of freedom for a plate model in X-Y plane

long **dofFree** = 0x00
no constraint (free node)

Records / structures

```
RNode = (  
double x    x coordinate of the node [m]  
double y    y coordinate of the node [m]  
double z    z coordinate of the node [m]  
long dof    nodal degrees of freedom  
)
```

Functions

long **Add** ([in] double **x**, [in] double **y**, [in] double **z**)
x *x coordinate of the node*
y *y coordinate of the node*
z *z coordinate of the node*
Adds a node to the model with dof = dofFree. If successful, returns node index otherwise returns an error code ([errDatabaseNotReady](#)).

long **AddWithDOF** ([in] double **x**, [in] double **y**, [in] double **z**, [in] long **dof**)
x *x coordinate of the node*
y *y coordinate of the node*
z *z coordinate of the node*
dof *nodal degrees of freedom*
Adds a node to the model with the specified degrees of freedom. If successful, returns node index otherwise returns an error code ([errDatabaseNotReady](#)).

long **Clear**
Deletes all nodes in the model. If successful, returns the number of deleted nodes otherwise returns an error code ([errDatabaseNotReady](#)).

long **Delete** ([in] long **Index**)
Deletes a node. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns Index otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **DeleteSelected**
Deletes all the selected nodes in the model. If successful, returns the number of deleted nodes otherwise returns an error code ([errDatabaseNotReady](#)).

long **IndexOf** ([in] double **x**, [in] double **y**, [in] double **z**, [in] double **eps**, [in] long **StartIndex**)
x *x coordinate of the node*
y *y coordinate of the node*
z *z coordinate of the node*
eps *tolerance for matching coordinates*
StartIndex *search begins at this node index*
 $1 \leq \text{StartIndex} \leq \text{Count}$
Finds a node by its coordinates. If successful, returns the node index otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errNotFound](#)).

long **GetConnectedLines** ([in] long **Index**, [out] SAFEARRAY(long)* **LineIdxList**)
Index *node index*
LineIdxList *line indices according to [AxisVMLines](#)*
Obtains indices of lines connecting to the node. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns the number of lines connecting to the node otherwise returns an

error code
([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **GetConnectedSurfaces** ([in] long **Index**, [out] SAFEARRAY(long)* **SurfaceIdxList**)

Index node index
SurfaceIdxList surface indices according to [AxisVMSurfaces](#)

Obtains indices of surface elements connecting to the node. $1 \leq \text{Index} \leq \text{Count}$.
If successful, returns the number of surface elements connecting to the node otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **SelectAll** ([in] BOOL **Select**)

Select selection state

If **Select** = True, selects all nodes.
If **Select** = False, deselects all nodes.
If successful, returns the number of selected lines otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

long **Count** number of nodes in the model
[RNode](#) **Node** [long **Index**] • a node by index
BOOL **Selected** [long **Index**] • gets or sets the selection status of a node
long **SelCount** number of selected nodes in the model

IAxisVMReferences

References in the model

Enumerated types

enum **EReferenceType** = {
 rtPoint = 0x01, reference point
 rtVector = 0x02, reference vector
 rtAxis = 0x03, reference axis
 rtPlane = 0x04, reference plane
 rtBeta = 0x05, reference angle
 rtNone = 0x06 }
Reference types

Records / structures

RPoint3d = (
double **x, y, z** x, y, z coordinates of a 3D point or components of a 3D vector [m]
)
RRefPoint = (
RPoint3d **P** reference point
)
RRefVector = (
RPoint3d **P1** first point of the reference vector
RPoint3d **P2** second point of the reference vector
)
RRefAxis = (
RPoint3d **P1** first point of the reference axis
RPoint3d **P2** second point of the reference axis
)
RRefPlane = (
RPoint3d **P1** first point of the reference plane
RPoint3d **P2** second point of the reference plane
RPoint3d **P3** third point of the reference plane
)

```

RRefBeta = (
double Beta    reference angle
)

RRefData = (
RRefPoint Point    reference point (rtPoint)
RRefVector Vector  reference vector (rtVector)
RRefAxis   Axis    reference axis (rtAxis)
RRefPlane  Plane   reference plane (rtPlane)
RRefBeta   Beta    reference angle (rtBeta)
)

RReference = (
EReferenceType ReferenceType  reference type
RRefData      ReferenceData  reference data
)

```

Functions

```

long Add ([in] RReference Item)
           Item    the reference

Adds a reference to the model. Reference data must be entered in the proper fields of
Item.ReferenceData depending on Item.ReferenceType.
If successful, returns the reference index otherwise returns an error code
(errDatabaseNotReady).

```

```

long Delete ([in] long Index)
           Index  the reference to delete

Deletes a reference. 1 ≤ Index ≤ Count.
If successful, returns Index otherwise returns an error code (errDatabaseNotReady,
errIndexOutOfBounds).

```

```

long DeleteSelected

Deletes selected references.
If successful, returns the number of deleted references otherwise returns an error code
(errDatabaseNotReady).

```

```

long IndexOf ([in] RReference Item)
           Item    the reference to find

Finds a reference by reference data. If successful, returns the index of the reference
otherwise returns an error code (errDatabaseNotReady, errNotFound).

```

```

long SelectAll ([in] BOOL Select)
           Select  selection state

If Select = True, selects all references.
If Select = False, deselects all references.
If successful, returns the number of selected references otherwise returns an error
code (errDatabaseNotReady).

```

Properties

```

long Count    number of references in the model
RReference Item [long Index] • reference by index
BOOL Selected [long Index] • gets or sets the selection status of a reference
long SelCount number of selected references in the model

```

IAxisVMSettings

Enumerated types

```

enum EPlaneToleranceType = {

```

ptRelativePerThousand = 0x01, *plane tolerance in per thousands*
ptAbsolute = 0x02 } *plane tolerance by value*
Plane tolerance type.

Records / structures

RPlaneTolerance = (
[EPlaneToleranceType](#) **ptType** *plane tolerance type*
double **Value** *(ptType = ptRelativePerThousand) plane tolerance is defined in per thousands of the biggest extension of the domain polygon*
(ptType = ptAbsolute) plane tolerance is defined as the maximum deviation from the domain plane [m]
)

Properties

Specifying invalid values triggers an error event (See [IAxisVMSettingsEvents](#))

double **CrossSectionEditingTolerance** •
Editing tolerance in the cross-section editor.

double **Editing Tolerance** •
Editing tolerance.

[RPlaneTolerance](#) **PlaneTolerance** •
Plane tolerance.

[ENationalDesignCode](#) **NationalDesignCode** •
The current national design code.

IAxisVMSurfaces

A modellben lévő felületelemek.

Enumerated types

enum **ESurfaceCharacteristics** = { **schLinear** = 0x00, **schTensionOnly** = 0x01,
schCompressionOnly = 0x02, **schBilinear** = 0x03 }
(not used)

enum **ESurfaceType** = {
stHole = 0x0, *hole*
stMembraneStress = 0x1, *membrane (plane stress)*
stMembraneStrain = 0x2, *tárcsa (plane strain)*
stPlate = 0x3, *plate*
stShell = 0x4 } *shell*
Types of surface elements.

Error codes

enum **ESurfacesError** = {
seLineCountCanBeOnly3Or4 = -100001 *a surface can have 3 or 4 edges only*
seCannotModify = -100002 } *cannot modify surface element properties*

Records / structures

double **RElasticFoundationXYZ** = (
x, y, z *elastic foundation stiffness in x, y, z directions [kN/m²]*
)

[ELineNonLinearity](#) **RNonLinearityXYZ** = (
x, y, z *nonlinear behaviour in x, y, z directions*
)

double **RResistancesXYZ** = (
x, y, z *resistance in x, y, z directions [kN/m²]*
)

double **RSurfaceAttr** = (
Thickness *surface thickness [m]*
[ESurfaceType](#) **SurfaceType** *surface type*
)

long	RefZId	reference index for the local z direction ($0 < \text{RefZId} \leq \text{AxisVMReferences.Count}$) or 0, if the reference is automatic
long	RefXId	reference index for the local x direction ($0 < \text{RefXId} \leq \text{AxisVMReferences.Count}$) or 0, if the reference is automatic
long	MaterialId	index of the surface material ($0 < \text{MaterialId} \leq \text{AxisVMMaterials.Count}$)
RElasticFoundationXYZ	ElasticFoundation	elastic foundation of the surface
RNonLinearityXYZ	NonLinearity	nonlinear behaviour of the elastic foundation
RResistancesXYZ	Resistance	resistance values of the elastic foundation
ESurfaceCharacteristics	Characteristics	nonlinear behaviour of the surface (not used)

Functions

long **Add** ([in] SAFEARRAY(long) **Linelds**, [in] long **DomainId**, [in] [RSurfaceAttr](#) **SurfaceAttr**)

Linelds indices of coplanar lines (3 or 4) forming edges of a surface.
See [AxisVMLines](#)

DomainId domain index if the surface element is part of a mesh
otherwise 0
 $0 \leq \text{DomainId} \leq \text{AxisVMDomains.Count}$

SurfaceAttr surface properties

Defines a new surface element.

If successful, returns the surface index otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seLineCountCanBeOnly3Or4](#) [*Linelds* contains less than 3 or more than 4 line indices]).

long **Delete** ([in] double **Index**)

Index index of the surface to delete

Deletes a surface element. $1 \leq \text{Index} \leq \text{Count}$.

If successful, returns *Index* otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **DeleteSelected**

Deletes selected surface elements.
If successful, returns the number of deleted elements otherwise returns an error code ([errDatabaseNotReady](#)).

long **SelectAll** ([in] BOOL **Select**)

Select selection state

If *Select* = True, selects all surface elements.
If *Select* = False, deselects all surface elements.
If successful, returns the number of selected elements otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

long **Count** number of surface elements in the model

[AxisVMSurface](#)* **Item** [long **Index**] surface element by *Index*

BOOL **Selected** [long **Index**] • gets or sets the selection status of a surface element

long **SelCount** number of selected surface elements in the model. Negative number is an error code ([errDatabaseNotReady](#)).

IAxisVMSurface

An AxisVM surface element object.

Enumerated types

enum **ESurfaceCharacteristics** = { **schLinear** = 0x00, **schTensionOnly** = 0x01, **schCompressionOnly** = 0x02, **schBilinear** = 0x03 }

(not used)

```
enum ESurfaceType = {  
    stHole = 0x0,           hole  
    stMembraneStress = 0x1, membrane (plane stress)  
    stMembraneStrain = 0x2, tárcsa (plane strain)  
    stPlate = 0x3,         plate  
    stShell = 0x4 }       shell
```

Types of surface elements.

Error codes

```
enum ESurfacesError = {  
    seLineCountCanBeOnly3Or4 = -100001    a surface can have 3 or 4 edges only  
    seCannotModify = -100002 }          cannot modify surface element properties
```

Records / structures

```
RElasticFoundationXYZ = (  
    double x, y, z           elastic foundation stiffness in x, y, z directions [kN/m2]  
    )  
RMatrix3x3 = (  
    double e11, e12, e13  
    double e21, e22, e23  
    double e31, e32, e33  
    )  
RNonLinearityXYZ = (  
    ELineNonLinearity x, y, z           nonlinear behaviour in x, y, z directions  
    )  
RResistancesXYZ = (  
    double x, y, z           resistance in x, y, z directions [kN/m2]  
    )  
RSurfaceAttr = (  
    double Thickness         surface thickness [m]  
    ESurfaceType SurfaceType     surface type  
    long RefZId              reference index for the local z direction  
                                (0 < RefZId ≤ AxisVMReferences.Count) or 0, if the reference is automatic  
    long RefXId              reference index for the local x direction  
                                (0 < RefXId ≤ AxisVMReferences.Count) or 0, if the reference is automatic  
    long MaterialId         index of the surface material  
                                (0 < MaterialId ≤ AxisVMMaterials.Count)  
    RElasticFoundationXYZ ElasticFoundation elastic foundation of the surface  
    RNonLinearityXYZ NonLinearity         nonlinear behaviour of the elastic foundation  
    RResistancesXYZ Resistance           resistance values of the elastic foundation  
    ESurfaceCharacteristics Characteristics nonlinear behaviour of the surface (not used)  
    )
```

Functions

```
long Modify (\[in\] SAFEARRAY(long) Linelds, \[in\] long DomainId,  
            \[in\] RSurfaceAttr SurfaceAttr)  
    Linelds   indices of coplanar lines (3 or 4) forming edges of a surface.  
                See AxisVMLines  
    DomainId  domain index if the surface element is part of a mesh  
                otherwise 0  
                0 ≤ DomainId ≤ AxisVMDomains.Count  
    SurfaceAttr surface properties  
  
Redefines a new surface element.  
If successful, returns the surface index otherwise returns an error code  
(errDatabaseNotReady, errIndexOutOfBounds, seLineCountCanBeOnly3Or4 [Linelds  
contains less than 3 or more than 4 line indices], seCannotModify).
```

Properties

```
double Area area of the surface element [m2]
```

SAFEARRAY(long)*	ContourLines	indices of surface edges (3 or 4)
long	DomainId	• domain index if the surface element is part of a mesh otherwise 0
RPoint3d	NormalVector	surface plane normal
RSurfaceAttr	SurfaceAttr	• surface properties
RMatrix3x3	TrMatrix	transformation matrix of the surface element
double	Volume	volume of the surface element [m^3]
double	Weight	mass of the surface element [kg]

IAxisVMSurfaceSupports

Surface supports of the model.

Records / structures

double	RStiffnessesXYZ	= (x, y, z elastic foundation stiffness in x, y, z directions [kN/m^2])
RLineNonLinearity	RNonLinearityXYZ	= (x, y, z nonlinear behaviour in x, y, z directions)
double	RResistancesXYZ	= (x, y, z resistance in x, y, z directions [kN/m^2])

Functions

long **AddSurfaceElasticFoundation** ([in] long **SurfaceId**, [in] [RStiffnessesXYZ](#) **StiffnessesXYZ**, [in] [RNonLinearityXYZ](#) **NonLinearityXYZ**, [in] [RResistancesXYZ](#) **ResistancesXYZ**)

SurfaceId	surface index $0 < \text{SurfaceId} \leq \text{AxisVMSurfaces.Count}$
StiffnessesXYZ	stiffnesses of the elastic foundation
NonLinearityXYZ	nonlinear behaviour of the elastic foundation
ResistancesXYZ	resistances of the elastic foundation

Adds a surface support (elastic foundation) to a surface element.

If successful, returns the index of the surface support otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **Delete** ([in] double **Index**)

Index surface support to delete

Deletes a surface support. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns Index otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **DeleteSelected**

Deletes selected surface supports. If successful, returns the number of deleted surface supports otherwise returns an error code ([errDatabaseNotReady](#)).

long **SelectAll** ([in] BOOL **Select**)

Select selection state

If **Select** = True, selects all surface supports.

If **Select** = False, deselects all surface supports.

If successful, returns the number of selected surface supports otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

long	Count	number of surface supports in the model
AxisVMSurfaceSupport*	Item [long Index]	a surface support object
BOOL	Selected [long Index]	• gets or sets the selection status of a surface support

long **SelCount** *number of selected surface supports in the model.*
Negative number is an error code ([errDatabaseNotReady](#)).

IAxisVMSurfaceSupport

An AxisVM surface support object.

Enumerated types

enum **ESurfaceSupportType** = {
sstSurfaceElasticFoundation = 0x00 } *elastic foundation of a surface*
Surface support type

Records / structures

RStiffnessesXYZ = (
double **x, y, z** *elastic foundation stiffness in x, y, z directions [kN/m²]*
)
RNonLinearityXYZ = (
[RLineNonLinearity](#) **x, y, z** *nonlinear behaviour in x, y, z directions*
)
RResistancesXYZ = (
double **x, y, z** *resistance in x, y, z directions [kN/m²]*
)

Properties

[RNonLinearityXYZ](#) **NonlinearityXYZ** • *nonlinear behaviour of the support*
[RResistancesXYZ](#) **ResistancesXYZ** • *resistances of the support*
[RStiffnessesXYZ](#) **StiffnessesXYZ** • *stiffnesses of the support*
[ESurfaceSupportType](#) **SupportType** *surface support type*

IAxisVMLine2d

A 2D line segment (straight line or arc) with orientation.

Enumerated types

enum **EArcAngleOrientation** = {
oClockwise = 0x00, *clockwise*
oCounterClockwise = 0x01 } *counterclockwise*
Arc or angle orientation

enum **ELine2dPointIndex** = {
piStart = 0x00, *kezdőpont*
piEnd = 0x01 } *végpont*
Point status.

enum **ELine2dType** = {
ItStraightLine = 0x00, *straight line*
ItCircleArc = 0x01 } *arc*
Geometry.

Records / structures

double **RPoint2d** = (
Coord1, Coord2 *2D coordinates [m]*
)

Functions

void **GetLinePoints** ([out] [RPoint2d](#) **StartPoint**, [out] [RPoint2d](#) **EndPoint**)
StartPoint *startpoint of the line*
EndPoint *endpoint of the line*

Obtains line endpoints.

void **SetLinePoints** ([in] [RPoint2d](#) **StartPoint**, [in] [RPoint2d](#) **EndPoint**)
StartPoint *startpoint of the line*
EndPoint *endpoint of the line*

Sets line endpoints

Properties

[RPoint2d](#) **CircleArcCenter** • (if *LineType* = *ItCircleArc*) center of the circle
[EArcAngleOrientation](#) **CircleArcOrientation** • (if *LineType* = *ItCircleArc*) orientation of the arc
[ELine2dType](#) **Line Type** • line type
[RPoint2d](#) **Point** [[ELine2dPointIndex](#) **Index**] • startpoint or endpoint of the line

IAxisVMPolygon2d

A list of [AxisVMLine2d](#) objects defining a closed two-dimensional polygon. A possible use of [IAxisVMPolygon2d](#) is to represent a custom cross-section shape. Polygon winding determines if the object is a hole or a boundary. A polygon is closed if the startpoint of the first line is the endpoint of the last one and the successive lines are connected. Line indices run from 1 to N.

Functions

long **AddLine** ([in] [AxisVMLine2d](#) **Line**)
Line *a new line*
Adds a line to the polygon.
If successful, returns the line index, otherwise an error code.

void **Clear**
Deletes all lines.

BOOL **DeleteLine** ([in] long **Index**)
Index *index of the line to delete*
Deletes a line from the polygon.
If successful, returns True, otherwise False.

Properties

BOOL **Hole** • if True, the polygon represents a hole (winding is clockwise). If False, the polygon represents a boundary (winding is counterclockwise). Setting the property reverse the lines and their order too.
[AxisVMLine2d](#)* **Line** [long **Index**] • a line in the polygon by index ($0 < \text{Index} \leq \text{LineCount}$)
long **LineCount** *number of polygon lines*

IAxisVMPolygon2dList

A list of [IAxisVMPolygon2d](#) objects. A possible use of [IAxisVMPolygon2dList](#) is to represent a complex cross-section including holes.

Functions

long **Add** ([in] [AxisVMPolygon2d](#) **Polygon**)
Polygon *a new polygon*
Adds a polygon to the list.
If successful, returns the polygon index, otherwise an error code.

void **Clear**
Deletes all polygons in the list.

BOOL **Delete** ([in] long **Index**)
Index *index of the polygon to delete*
Deletes a polygon from the list.
If successful, returns True, otherwise False.

Properties

long **Count** *number of polygons in the list*
[AxisVMPolygon2d](#)* **Item** [long **Index**] • *a polygon in the list by index ($0 < \text{Index} \leq \text{Count}$)*

IAxisVMLines3d

A list of three-dimensional line segments. IAxisVMLines3d can represent mesh-independent load polygons or polylines.

Enumerated types

enum **EArcAngleOrientation** = {
oClockwise = 0x00, *clockwise*
oCounterClockwise = 0x01 } *counterclockwise*
Orientation or an arc or angle if seen from a direction opposite to the plane normal.

enum **ELine3dType** = {
ItStraightLine3d = 0x00, *straight line*
ItCircleArc3d = 0x01 } *arc*
Line geometry.

Records / structures

RLine3d = (
[ELine3dType](#) **LineType** *line geometry*
[RPoint3d](#) **P1** *startpoint*
[RPoint3d](#) **P2** *endpoint*
[RPoint3d](#) **ArcCenter** *(if LineType = ItCircleArc3d) center of the circle*
[EArcAngleOrientation](#) **ArcOrientation** *(if LineType = ItCircleArc3d) orientation of the arc*
[RPoint3d](#) **NormVect** *(if LineType = ItCircleArc3d) arc plane normal*
)

RPoint3d = (
double **x, y, z** *x, y, z coordinates of a 3D point or components of a 3D vector [m]*
)

Functions

long **Add** ([in] [RLine3d](#) **Item**)
Item *a new line*
Adds a line to the list.
If successful, returns the line index, otherwise an error code.

void **Clear**
Deletes all lines in the list.

BOOL **Delete** ([in] long **Index**)
Index *index of the line to delete*
Deletes a line from the list.
If successful, returns True, otherwise False.

Properties

long **Count** *number of lines in the list*
[RLine3d](#) **Item** [long **Index**] • *a line of the list by index ($0 < \text{Index} \leq \text{Count}$)*

Event handling

The usual method to build a COM application is that a COM client calls the functions of the COM server. Events make it possible for the COM server to call functions of the COM client to send notifications or error messages. In order to handle these events the client has to implement a standard interface receiving server calls. This interface is called event sink.

The list of AxisVM COM event interfaces:

IAxisVMApplicationEvents

Events:

void **Loaded**

When COM server is created it checks if AxisVM application has to be launched or not. If AxisVM is not running it is launched. Loaded is called when AxisVM is loaded. Loaded will never be called if AxisVM is already running when the COM server starts.

IAxisVMModelsEvents

Események:

void **Cleared**

All elements of the list has been cleared.

void **Deleted** ([in] long **Index**)

Index *index of the list element to delete*

An element of the list has been deleted.

void **Error** ([in] long **ErrorCode**)

ErrorCode *the error code*

Called in case of an error.

IAxisVMCrossSectionsEvents

See [IAxisVMModelsEvents](#)

IAxisVMDomainsEvents

See [IAxisVMModelsEvents](#)

IAxisVMLinesEvents

See [IAxisVMModelsEvents](#)

IAxisVMLineSupportsEvents

See [IAxisVMModelsEvents](#)

IAxisVMLoadCasesEvents

See [IAxisVMModelsEvents](#)

IAxisVMLoadCombinationsEvents

See [IAxisVMModelsEvents](#)

IAxisVMLoadGroupsEvents

See [IAxisVMModelsEvents](#)

IAxisVMLoadsEvents

See [IAxisVMModelsEvents](#)

IAxisVMMaterialsEvents

See [IAxisVMModelsEvents](#)

IAxisVMNodalSupportsEvents

See [IAxisVMModelsEvents](#)

IAxisVMNodesEvents

See [IAxisVMModelsEvents](#)

IAxisVMReferencesEvents

See [IAxisVMModelsEvents](#)

IAxisVMSettingsEvents

Események:

```
void Error ([in] long ErrorCode)  
           ErrorCode the error code  
           Called in case of an erroneous setting.
```

IAxisVMSurfacesEvents

See [IAxisVMModelsEvents](#)

IAxisVMSurfaceSupportsEvents

See [IAxisVMModelsEvents](#)

AxisVM COM plugin 1.0

Any external EXE program can create a COM client controlling the AxisVM COM server. Another way to write automation extensions is to build a DLL plugin for AxisVM.

When AxisVM is launched it looks for *.DLL files in the <AxisVM installation folder>\plugins folder. Plugins are simple DLL files exporting certain functions:

C syntax:

```
unsigned int32 GetPluginVersion;  
unsigned int32 GetMenuItemTextA(const void *Buffer, unsigned int32 BufferSize);  
unsigned int32 GetMenuItemTextW(const void *Buffer, unsigned int32 BufferSize);  
void SetAxisVMMainFormHandle(const unsigned int32 FormHandle);  
void OnMenuItemClick(void);
```

Pascal syntax:

```
function GetPluginVersion : DWORD; stdcall;  
function GetMenuItemTextA(const Buffer: Pointer; BufferSize: DWORD) : DWORD; stdcall;  
function GetMenuItemTextW(const Buffer: Pointer; BufferSize: DWORD) : DWORD; stdcall;  
procedure SetAxisVMMainFormHandle(const FormHandle: THandle); stdcall;  
procedure OnMenuItemClick; stdcall;
```

If AxisVM finds a DLL in the above plugin folder exporting these functions and **GetPluginVersion** returns the appropriate value a new menu item with the plugin name is automatically created in the *Plugins* menu of AxisVM. Plugin can be started clicking this menu item.

Functions to export

- | | |
|---------------|--|
| unsigned long | GetMenuItemTextA ([out] void* Buffer , [out] unsigned long BufferSize)
Buffer <i>plugin name</i>
BufferSize <i>maximum number of characters in the name</i>
<i>Copies the plugin name into the Buffer. The name must contain 8-bit ANSI characters. Returns the actual length of the plugin name.</i> |
| <hr/> | |
| unsigned long | GetMenuItemTextW ([out] void* Buffer , [out] unsigned long BufferSize)
Buffer <i>plugin name</i>
BufferSize <i>maximum number of characters in the name</i>
<i>Copies the plugin name into the Buffer. The name must contain 16-bit Unicode characters. Returns the actual length of the plugin name.</i> |
| <hr/> | |
| unsigned long | GetPluginVersion
<i>Returns the plugin version (should be 1).
If this function returns some other value the plugin does not appear in the AxisVM Plugins menu.</i> |
| <hr/> | |
| void | OnMenuItemClick
<i>This function is called when the user clicks the plugin menu item. Plugin is executed in a separate thread.</i> |
| <hr/> | |
| void | SetAxisVMMainFormHandle ([in] unsigned long FormHandle)
FormHandle <i>the handle of the AxisVM main window</i>
<i>Call this function to retrieve the handle of the AxisVM main window.</i> |